ETH zürich IBM Research | Zurich

Master Thesis

Implementing and Benchmarking a Floquet Code on Superconducting Hardware

---

Quantum Computational Science Group

IBM Quantum, IBM Research Europe - Zürich

Department of Information Technology and Electrical Engineering (D-ITET) & Department of Physics (D-PHYS)

ETH Zürich

Ian Hesner

October 2023

Supervisors

Dr. James Wootton

Dr. Joseph Renes

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Implementing and Benchmarking a Floquet Code on Superconducting Hardware

Authored by (in block letters):

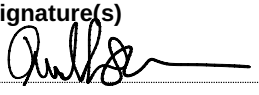
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Hesner	Ian

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 2023.10.27	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Abstract

Floquet codes are a novel style of quantum error correcting code with dynamically-generated logical observables defined on a hexagonal lattice. The hexagonal layout makes it an excellent candidate to fit on the heavy-hex layout of IBM Quantum devices. In this work, we view the Floquet code through the anyonic model of topological quantum computing to define a planar implementation that fits on existing devices. Actually running the Floquet code on IBM's `ibm_sherbrooke` Eagle device shows that current devices are still well above Floquet code thresholds. Further work investigated a benchmarking technique to more accurately compare simulation data to real device data. Typically, simulation results for running quantum error correcting codes compare quite poorly to results obtained from real devices. Average detector triggering likelihoods were explored to be used as an intermediate parameter to directly compare the two. With this as an intermediate parameter, an effective-p value could be derived for a real device, comparing the complex noise landscape of a real device to a flat noise model defined by a single parameter p. This effective-p value does a much better job of simulating real device performance over other standard Pauli noise model techniques. It was found that for the best locations that can fit a Floquet code on `ibm_sherbrooke`, it has an effective-p value of just over 2%.

Contents

1 Introduction

Topological quantum computing was originally defined by Alexei Kiteav, where he showed a two-dimensional quantum system with anyonic excitations can be considered as a quantum computer [Kit03]. In his work on the honeycomb lattice model, Kitaev showed that it can also be defined on a hexagonal lattice. [Kit06]. This construction was mostly ignored for many years because the resulting toric code defined on this hexagonal lattice was believed to not contain any logical operators from a subsystem code perspective [Pou05]. That all changed in 2021 when Hastings and Haah showed, through a particular measurement scheduling, logical operators can be defined on the hexagonal lattice [HH21]. These logical operators were quite unique. Historically, logical operators of any code were defined statically. The logical operators created by Hastings and Haah were dynamically generated, meaning they changed location and Pauli type throughout the dynamics of the code. As a result, there has been a lot of exploration in the past few years into what are now referred to as Floquet codes, because of this new style of dynamically generated logical operators [GNM22; Gid+21; HH22; Kes+22; Pae+23; SBB23; ESD23; Aas+23; Woo22a]. To date, nobody has run a full Floquet code on hardware. The closest was when James Wootton showed this style of plaquette operator used in these Floquet codes works on actual hardware by demonstrating the operation of a single plaquette on an IBM Quantum device [Woo22b]. This thesis attempts to run these Floquet codes on IBM Quantum's devices, which feature a Heavy-Hex layout of superconducting qubits. While it is clear that current devices are not at sufficiently low noise levels to run a Floquet code below threshold, a benchmarking technique using the average detector likelihood of a run was developed to better understand the performance of a device, and perform more accurate simulations of real devices.

The body of this work starts off in Section 2 by reviewing the anyonic model

of topological quantum computing. This is done because it is much easier to understand the dynamics of the Floquet code through this anyonic model. Review of the model is done using the standard implementation of the surface code, here referred to as the Vanilla Surface Code (VSC) because it is the most familiar and well understood code in the field. We do this review using the VSC because we will see that the Floquet code is just another surface code. A brief Section 3 goes over how matching decoding corrects logical errors through the anyonic lens. Section 4 defines the Floquet code on torus to gain an understanding of how the code works in the bulk, while Section 5 flattens the code into a planar implementation that will be run on real devices. The results of running the code on real devices is shown in Section 6, and shows that current devices are well above threshold. Also included in Section 6 is a discussion of the new detector likelihood benchmarking technique that is shown to lead to better simulation of real devices than current standard Pauli noise model techniques. Appendix A also gives a brief overview of ZX calculus for Quantum Error Correction, because there are ZX diagrams using Pauli webs used in the main text. This appendix will explain how to read such diagrams for those unfamiliar with ZX calculus and Pauli webs.

2 Anyonic Model of Topological Quantum Computing

Explaining the nuances of the Floquet code becomes much simpler through an anyonic point of view. Because of that, we will first review the basics of anyonic topological quantum computation to define the language that is simplest to explain the Floquet code. The Vanilla Surface Code (VSC) will be used as an example because it is more well known to those familiar with quantum error

correction.

2.1 Anyon Basics

The original system described by Kiteav was defined on a square lattice with data qubits on the edges and different types of parity checks on each vertex and face. This representation can be seen as the 2D matching graph in the decoding section, but to stay grounded in actual implementations we will look at the layout in Figure 1 instead of the lattice, as they represent the same thing. The layout shows blue and red plaquettes, with data qubits in the corners. Each plaquette represents a weight-4 parity check of the surrounding qubits. In physical systems this is done by having an ancilla in the middle of the plaquette and using CNOTs to entangle it to the four qubits in the corners before measuring. The different color plaquettes represent two separate types of checks, Pauli Z-checks for the red plaquettes and Pauli X-checks for the blue ones.

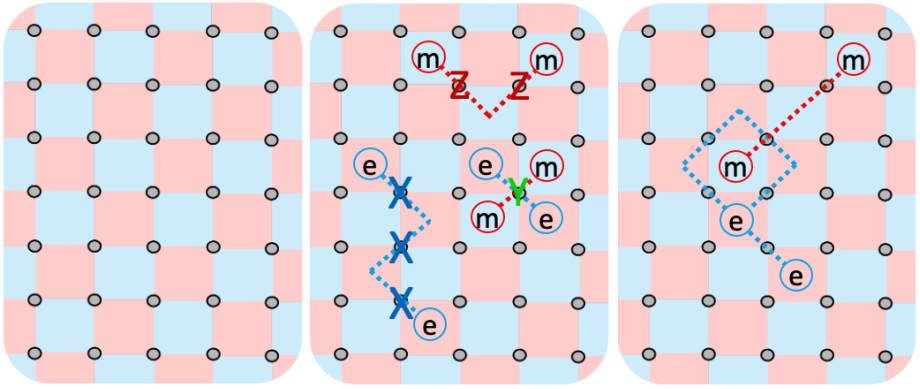


Figure 1: Above are three panels showing a section of the bulk in a VSC. The left most panel shows the layout, with red weight-4 Z-type checks, blue weight-4 X-type checks, and gray data qubits. The left most panel is considered to be without any excitations. The center panel shows the occurrence of Pauli error strings on the data qubits, and how the resulting anyons are created as a result. The right most panel shows a pair of error strings performing a braiding operation, where an e-type anyon is moved around an m-type anyon.

A very convenient fact about quantum error correcting codes constructed with Pauli checks is that we only need to consider Pauli noise when ignoring leakage. Any single qubit unitary can be broken down as $U = \alpha\mathbb{I}+\beta X+\gamma Y+\delta Z \mid \alpha, \beta, \gamma, \delta \in \mathbb{C}$, where X, Y, and Z are the Pauli matrices. As a result, any arbitrary noise on a single qubit can be broken down into Pauli noise. Upon measuring the adjacent checks, which detect if a Pauli channel has been applied to any of the adjacent qubits in odd parity, the noise gets projected down to pure Pauli noise. For the VSC layout, this means for any X(Z)-error will cause a Z(X)-type red(blue) detector to trigger. These state changes will be interchangeably referred to as detection events or excitations. Because a Y-error can be decomposed into an X- and Z-error up to a phase, it creates two pairs of detection events, one pair for each check type. Because the noise breaks down like this, we can consider the two detector types as completely separate.

These pairs are the smallest examples of string operators. String operators are operators comprised of one or more Pauli operators resulting in a pair of excitations at the ends of the string. We define excitation color by the type of error that caused their creation. This results in an anyon being created on the opposite color of plaquette. Except in special cases, both excitations on either side of the string are of the same type. Some examples of string operators can be seen in Figure 1b.

Figure 1c shows what is referred to as a braiding operation. A braiding operation is when an anyon is moved around an anyon of the opposite type. The image shows an e-anyon moving around an m-anyon and returning to the same location. When it does this it picks up a phase of (-1). This special braiding behavior is what defines the group structure as abelian anyons, whose properties are rooted in quantum field theory. In fact, this phase change can be interpreted as the Aharonov-Bohm effect, and these abelian anyons are known to exist in

many physical system [Kit03]. Although they will not be covered here, non-abelian anyons can also be created by the use of code deformations. Existence of these non-abelian anyons has been shown experimentally on a superconducting qubit device, giving some experimental verification to this anyonic model [And+22].

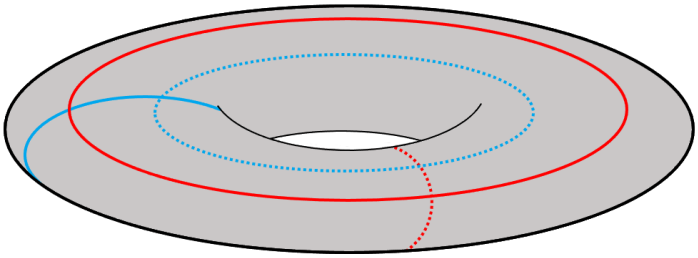


Figure 2: A visualization of a torus containing quantum information. A toric code contains two logical qubits, with their logical operators wrapping around the torus in opposite directions. The two qubits are shown with solid and dashed lines. To form a logical qubit, the two opposite anyon types must cross once to ensure the proper anti-commutation relations. The crossings of the two different logical qubits only happen between two logical operators of the same type, meaning they commute and to not effect each other.

Topological quantum computing stores quantum information by using these anyons' characteristic behaviors. Consider the lattice where these anyons live to be put on a torus. If a string operator is able to wrap around a torus, annihilating its excitations with each other on the other side, then it becomes a string operator with a phase, but no excitations. These silent string operators are the logical operators that form a logical qubit defined on the torus. Two logical qubits are able to be encoded in the standard toric VSC. Figure 2 shows how these two pairs of logical operators live on the torus. Notice that the solid (and dashed) logical operators have the two opposite type operators crossing each other. This ensures the proper anti-commuting relationship expected from X- and Z-logical operators of a qubit. Any other crossings between different logical qubits are of the same Pauli type, so they commute and do not affect

each other. This is the basic setup for how logical operators exist on a torus viewed through an anyonic model.

2.2 Planar Anyonic Codes

The planar VSC was first defined in [Den+02] and then rotated in [TS14] for a more efficient implementation. The rotated planar surface code has been at the center of quantum error correction research ever since. It was the first topological code to be implemented on superconducting qubit based devices by QuDev [Kri+22] and Google AI [Ach+22] in the past couple of years. In Figure 3, the left layout shows how the rotated planar surface code looks with proper boundaries so it fits on a 2D chip. It has the same bulk as in the toric case, but with weight-2 truncated plaquettes along the borders.

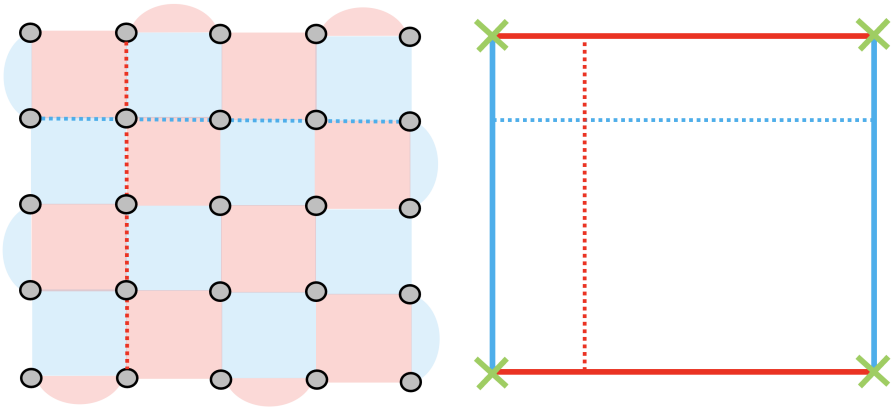


Figure 3: The left image shows a planar layout of the VSC. The boundaries are capped with weight-2 truncated plaquettes to achieve the correct condensation. Because blue anyons will condensate on the left and right borders, and red anyons will condensate on the top and bottom borders, we use those colors to create a new abstract view of the surface code on the right. The bulk is not shown here, but instead just the main features of the code: the condensation boundaries, logical operators, and Y-detects. The Y-defects in the corners shown with a green X are able to condensate both anyon types and act an non-abelian anyons.

Notice that the top and bottom borders have a different type of truncated

plaquette type than the left and right borders. The type of truncated plaquette used to create these borders affects what type of border is created. The term "condensation" is used to refer to when an anyon associated with a string operator can be absorbed into a border. These borders are designed to condensate one type of string operator, but not the other. Looking at Figure 4, it can be seen how each type of string operator interacts with each border type. Adding truncated plaquettes of only one anyon type blocks the condensation of that same type by offering a location at the boundary for the excitations to live. This strategy gives the control to determine what type of anyons condensate where.

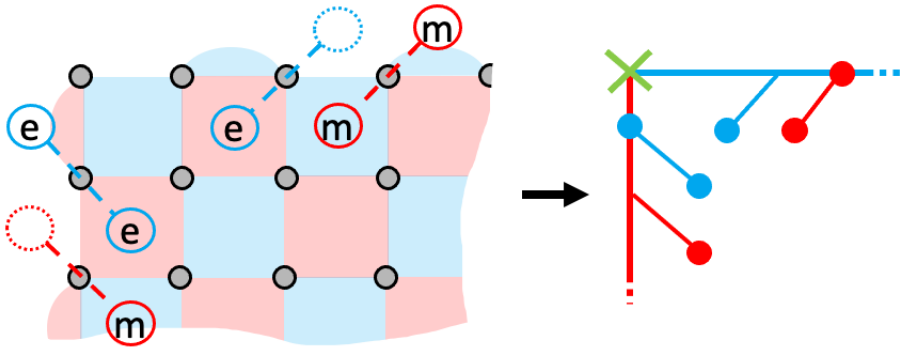


Figure 4: This image is intended to show the microscopic details of anyon condensation. A pair of anyons of each type is created on each style of border. It can be seen only one type of anyon creates a full pair of anyons. The other type is missing a plaquette where the other anyon would be created, shown with a dotted circle. The left image shows how these condensation events look like in the abstract picture defined in the previous figure.

This allows for more abstract pictures of these codes when thinking through the anyonic model. The images shown on the right of the two figures in this section show such abstract pictures that only include borders, excitations, and string operators. The borders show the color of the anyon type it condenses. There are also green Xs in each corner. These Xs represent Y-defects where both anyon types can condense in the corner. The Y-defects play an important

role in logical operations, and are a form of non-abelian anyons that can be braided to perform logical operations [Bro+17; FG19; Gid23; Bom+23a].

In the toric case, we saw a pair of orthogonal logical operators that make up a logical qubit wrap around the toric surface in topologically different directions, ensuring they cross once to ensure anti-commuting. In the planar case, logical operators stretch from boundary to boundary across the code, condensating on each side. The borders must be opposite each other so that logical operators maintain a minimum distance, ensuring larger codes protect against logical errors more robustly. Setting up the borders like this also ensures that logical operators must cross each other somewhere in the bulk so that the anti-commutation relation is preserved. Any string operator that condensates on opposite boundaries is a logical operator, because every path linking opposite boundaries is equivalent up to the inclusion of stabilizers. When defining the floquet code, we will arrive back at this exact same picture because the floquet code and VSC are just variations of the same abstract anyonic model of planar surface codes.

3 Matching Decoding for Surface Codes

Currently the most common decoding method for topological quantum computing is done with matching algorithms. The current preferred method is using Minimum Weight Perfect Matching (MWPM) [FWH12; Fow14] for its high accuracy. The most competitive alternative decoding methods are simply approximate MWPM algorithms that are designed to run faster, such as Union Find and its variants [DN21; WLZ22; Del20; Liy+23]. In this section, we will discuss the general matching process, not going into specifics of individual algorithms, in the context of the anyonic model seen, and linking it to the underlying group theory.

Any quantum error correcting code encodes logical information within a subspace of the Hilbert Space H made up by the data qubits of the system. Recall the surface code projects any two-level unitary noise into Pauli noise, ignoring leakage. This means any noise channel can be considered to be within the Pauli group $P_n = \langle i\mathbb{I}, X_1, Z_1, ..., X_n, Z_n \rangle$ for a system with n data qubits, where X_i and Z_i are Pauli channels acting on the ith qubit. The string operators discussed before that result in a pair of excitations live inside P_n. Examples of Pauli strings $\in P_n$ and their possible corrections are shown in Figure 5.

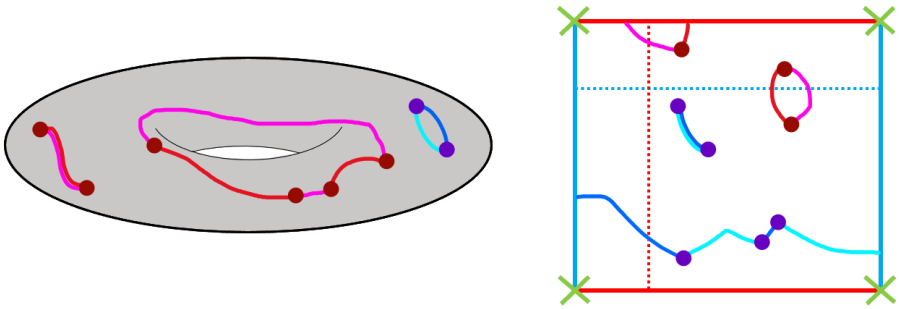


Figure 5: Examples of decoding on a toric and planar code. Red and blue represent different anyon types, while dark and light shading represent error strings and decoding strings. Both codes show three cases. First is perfect decoding, where light and dark are right on top of each other. This is the ideal case for decoding and returns the state exactly to where it started. Second is when decoding does not decode the exact path, but despite that, only forms a trivial loop. These loops are okay because they are just products of stabilizers, which commute with the logical operator. This can easily be seen in the planar case with the dotted blue logical operator crossing the small red loop in the upper right. There is also a loop formed with the border at the top, which acts the same. Because it crosses the strings twice, the logical operator's total parity remains unchanged. Lastly are the logical errors. In the toric case, a logical error is shown in red looping around the torus, and in the planar case a blue string condensating on opposite blue boundaries. It is easy to see in the planar case that it crosses the red logical operator, flipping its value.

What matching does is pair each detection event to either another detection event or a valid boundary. The path that it used to match depends on the lowest weight route in the matching graph. In Figure 5, the light shaded lines

represent possible matchings that would be associated with the given set of syndromes caused by the dark shaded Pauli error strings. The matchings are not always perfect. Applying the decoded string operators results in the same set of detection events that were detected initially. As a result, all of the anyons annihilate each other when the decoding string operators are applied. This results in the system returning to the +1 eigenspace of the stabilizers with no excitations present. At the group theory level, what these matchings are doing is returning the system to the $N(S)$ subspace, where S is defined as the group generated by the stabilizers (our weight-4 checks). This subspace is the normalizer of the stabilizers, and is defined as $N(S) = \{g \in G \mid gs = sg, \forall s \in S\}$. Note the logical operators are a part of this subspace by definition since logical operators must commute with all stabilizers [Got97]. A physical way of thinking about this when matching corrections are applied, all of the excitations cancel out and the space returns to the +1 eigenspace of S.

If errors are not decoded correctly, following paths that were not the original error strings, they either form a loop of stabilizers or a logical operator in the matching graph. This must be the case because $N(S) = \langle S, \hat{X}_L, \hat{Z}_L \rangle$. Topologically trivial loops, loops that do not wrap around a torus, do not affect the state. Mathematically the error strings that form loops are always just products of stabilizers, which by definition all commute with the logical operators and do not affect the state of the logical qubit. Visually it is easy to view a loop as not affecting a logical operator because a logical operator must cross a loop twice to pass through it, and as a result its overall parity will remain unchanged. If there exists a string from boundary to boundary or around the torus, that means a logical error occurred. Figure 5 shows examples of how decoding can result in remaining Pauli strings while still in N(S). Each code includes an example of a logical error string, which would flip the opposite logical

observable. From a group theoretic perspective, what matters is what element of the Quotient group $N(S)/S$ decoding pushes the space into. What this quotient group does is groups the decoded space down in four equivalence classes that only differ by the presence of logical operator Pauli strings. Put differently, $N(S)/S \simeq \langle \hat{X}_L, \hat{Z}_L \rangle$, and all that matters at the end of the decoding process is if a logical operator occurred.

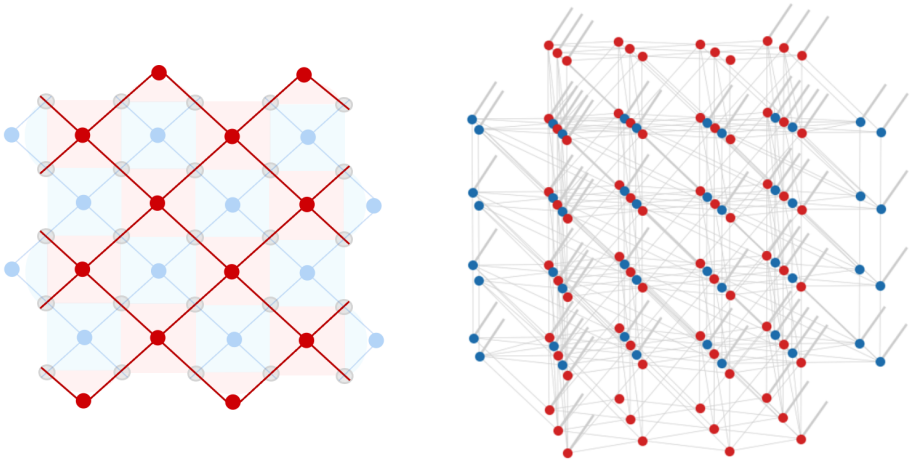


Figure 6: To the left, the red graph shows the original square lattice for a rotated planar VSC. The layout seen before is faded and added underneath to show the correspondence. Data qubits are defined to be on the edges of the lattice, with primary checks at the intersections. The lattice was originally defined as having the dual checks be defined on the faces. Just for clarity, a faded blue dual lattice is also shown to see how it lines up with the faces of the primary lattice. Matching decoding would work on this single 2D graph if measurements were perfect, but in reality that is not the case. To account for measurement errors, the graph must be extended into 3D as shown on the right, with boundary edges shown in thicker gray floating up and to the right. The graph gains vertical edges that capture measurement errors, as well as diagonal edges for errors that occur between CNOT gates in a given measurement round.

The graph that matching occurs on is not the original lattice that the code is defined on. For a single measurement round with perfect measurements this is the case, but in reality measurements are imperfect. To account for this, the matching graph must be extended into the third dimension to account for

measurement errors that can occur. Figure 6 shows how a flat matching graph is extended into the time dimension as well. Despite the changes in the matching graph, the previous intuitions hold. Matching sends the state to $N(S)$, loops present after decoding do not affect the logical state, while strings attaching opposite borders or wrapping around the torus result in logical errors.

4 Floquet Codes

What is now referred to as the Floquet code was originally developed by Hastings and Haah in [HH21] and lived on a torus. It is defined on a Honeycomb lattice as shown in Figure 7 that was originally defined by Kitaev in [Kit06]. For a long time the Honeycomb lattice was considered to not contain any logical operators, which is true when viewed as a subsystem code [Pou05]. When peeling back the time dynamics of the specific measurement cycle that Hastings and Haah defines, logical qubits able to store quantum information can be found to be dynamically-generated.

Figure 7 shows this Honeycomb layout, where the data qubits are pictured in grey with three connections and each connection is of a different color. This leads to no links of the same color attaching to the same data qubit. Defining the coloring like this also allows to give the interior plaquette space a color as well. Every plaquette color is surrounded by links of purely different colors. Taking red as an example, this has the effect of causing every red plaquette to be surrounded by blue and green links and have red links going away from it in support.

In this colored lattice, each link has an implied ancilla qubit on it. This is because they represent weight-2 parity check, which needs a mediating ancilla qubit. This results in an overall heavy hex layout that is seen in the larger IBM superconducting qubit chips. In this section we associate color with Pauli

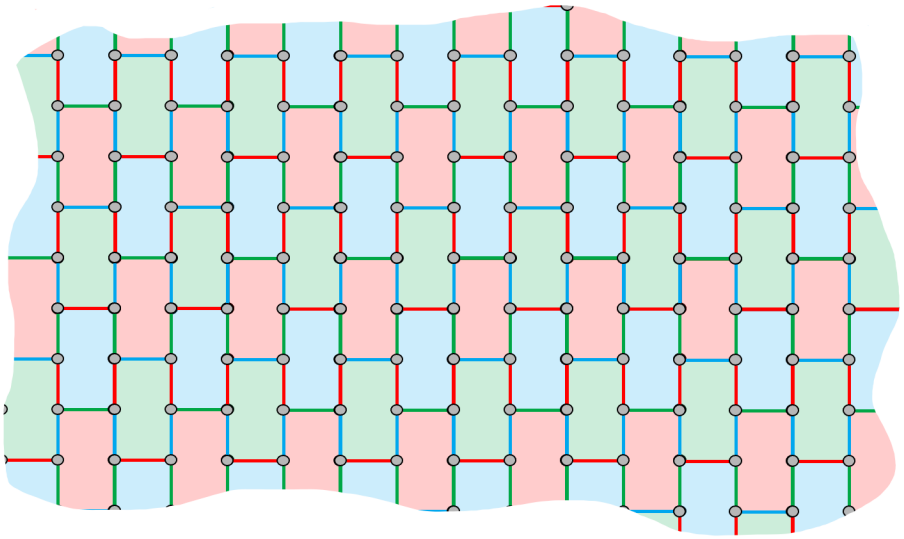


Figure 7: The Floquet code is defined on a hexagonal lattice with RBG coloring as shown in the image. Each gray dot represents a data qubit associated with three link operators. Each data qubit has one link operator of each color. For actual systems, there is an implicit ancilla qubit in the center of the link operator to entangle with the two adjacent data qubits before measurement. For the plaquette coloring, each plaquette is defined to commute with all of the link operators around it. Here we associate Red, Blue, and Green with Z-, X-, and Y-Pauli types. In the example of a green plaquette, this means it can be created by multiplying the six link operators around it together. Since at every data qubit, a Red Z-link and Blue X-link are multiplied together, in total it createds a weight-6 Y-operator.

type. In the original Hastings Haah paper, a check operator's Pauli type was associated with the orientation of the check operator. Other papers that have built upon the Floquet work have shown there is a flexibility in Pauli association [GNM22; Kes+22]. We will show these are all anyonically the same. For this section, red links will be defined as ZZ-parity checks, blue as XX-parity checks, and green as YY-parity checks.

The measurement scheduling for this code is defined as red, blue, green, red, blue, green, etc. The overall code is very sensitive to this measurement scheduling. One of the main points of this section is to analyze the Floquet

code through this measurement scheduling to understand the dynamics of this code, then use anyonic reasoning to change this measurement schedule. When changing this scheduling, all of the detectors and logical operators also change. The overall character of the code remains the same on a macroscopic level, but the details on a microscopic level can very drastically. The dynamics of this code are able to be defined between measurement rounds as Instantaneous Stabilizer Groups (ISGs). Every measurement round projects the code into the next ISG, redefining what plaquettes are relevant and the logical operators' location and Pauli type as they move through the code. The ISGs are cyclic, with a period of six. Although the color repeats every three rounds, the logical operator and detector dynamics take double that to fully reset. The next subsections will look at how detectors and the logical operators evolve through the ISGs, then put it all together into the previous anyonic perspective.

4.1 Floquet Plaquettes

To start dissecting this code, lets look at a single plaquette detector microscopically. Plaquette detectors are defined by multiplying in the edges around the plaquette together. Using a blue plaquette as an example, it has a border made up of three red ZZ-checks and three green YY-checks. While each of these checks individually do not commute with each other, the product of all three do commute with each other. This allows for the overall measurement of the product of all six link operators to be deterministic in terms of overall parity. From a stabilizer formalism perspective, these plaquettes can be viewed as stabilizers. In the case of the blue plaquette, all links multiplied together will create a weight-6 hexagonal X-type stabilizer. When a blue measurement round comes around, the links measure the blue XX-checks, which commute with this X-type stabilizer, so comparing stabilizers parity in subsequent rounds is de-

terministic. To create a full detector from this, the links on either side of the blue measurement round must be compared, just like stabilizer measurements are combined to create syndromes in the typical VSC.

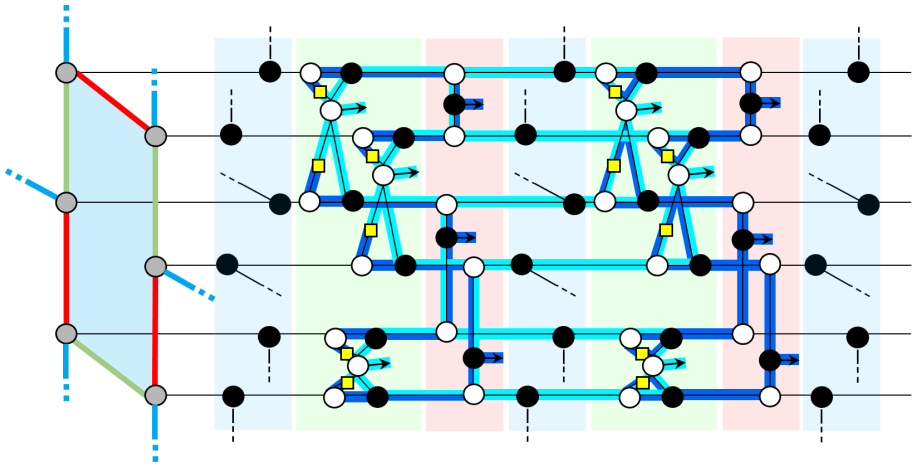


Figure 8: Shown here is a ZX diagram of a single plaquette, and a Pauli web associated with a single detector overlaid. The layout to the left shows how the data qubits' represented with horizontal lines are connected. For details on how to read these diagrams, see Appendix A. It can be seen that for a single plaquette detector, two measurement rounds of links need to be multiplied together to create the desired X-type when growing the detector, and similarly when collapsing. When expanding the time dynamics like this, it shows how viewing this detector as an X-type detector misses a lot of nuance. Ultimately, the detector only spends half of its lifetime as an X-type detector, when at other times it acts as a different Pauli type.

The stabilizer formalism point of view can be a bit over simplistic for this code. It misses some of the details that live within the specific time dynamics. Figure 8 shows a ZX diagram of a single plaquette in the Honeycomb lattice for the defined the measurement scheduling. This figure shows a Pauli web overlaying the ZX diagram, where the Pauli web represents a detector. For a more in depth explanation of how to read these diagrams, see Appendix A. From the Pauli webs, it can be seen that there are three distinct periods of different Pauli type detecting regions for the detector spanning four different ISGs. The

evolution goes as follows:

- Measure Y: Expanding detector, results included in detector
- ISG: Y-type detection region
- Measure Z: Changing detection Pauli type, results included in detector
- ISG: X-type detection region
- Measure X: Commutes with plaquette, results do not affect detector
- ISG: X-type detection region
- Measure Y: Changing detection Pauli type, results included in detector
- ISG: Z-type detection region
- Measure Z: Collapsing detector, results included in detector

The other colored plaquettes exhibit the same behavior, but with different Pauli evolutions.

4.2 Logical Operator

One of the most unique parts of the Floquet code is how the logical observables act. Its is the first example of a code where the logical observables are dynamic, changing through the ISGs. Figure 9 shows one logical operator wrapping around a torus through a full measurement cycle of all six ISGs. The location of the logical operator is defined as a loop around the torus, with both logical operators wrapping around different directions, just like with the VSC. Once this path is defined, for each measurement round the link operators that are measured along the path must be multiplied into the logical operator. This has the affect of changing the Pauli type as well as the location of the logical

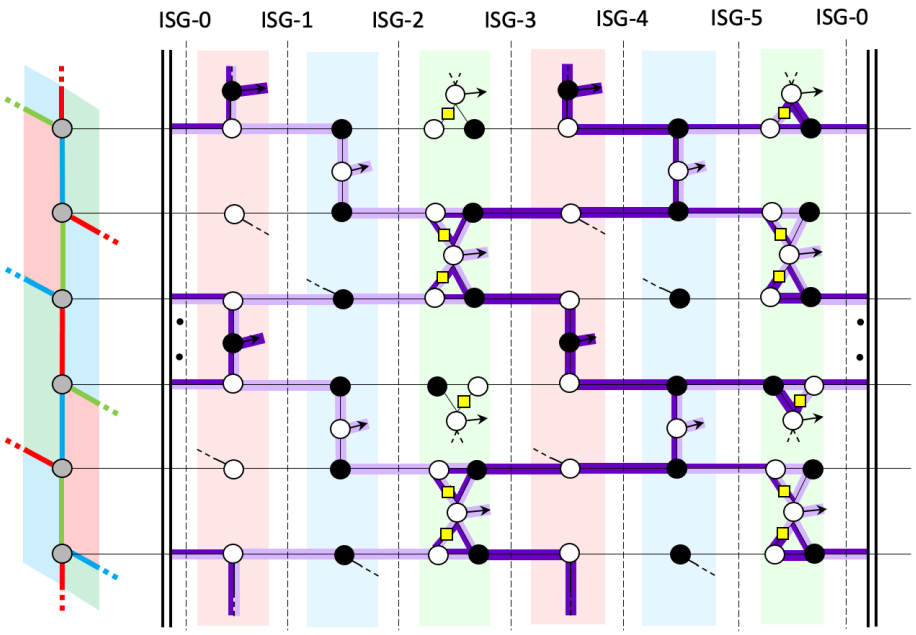


Figure 9: Above is a ZX diagram with a Pauli web overlay representing a logical operator. This logical operator lives on a torus, and in the layout to the left of the image, the top and bottom red link operators represent the same operator looping around. The entire repeated six stage ISG cycle is shown, which gets repeated throughout the code. Each ISG is marked so it can be seen how the logical operator is defined for each stage. Notice how in the Pauli = color measurement cycle definition, every link that is included in the loop around the torus is multiplied into the logical operator every round, either moving the logical operator or changing its Pauli type.

operators from round to round. It can be seen that each time a check operator is multiplied into the logical observable, it is necessary to avoid commutation issues in the next measurement round.

4.3 Anyonic Evolution Through ISGs

Lets try and put these evolving detectors and logical operators into the context of the anyonic model from Section 2. In the VSC, the anyon type was tied to Pauli type. In the Floquet code as defined above, that is no longer the case. But because each logical operator is directly linked to one type of anyon, we can use

Plaquette Color	ISG-0	1	2	3	4	5	Anyon Type
Red	-	-	X	Z	Z	Y	e
Red	Z	Z	Y	-	-	X	m
Blue	Y	X	X	Z	-	-	e
Blue	Z	-	-	Y	X	X	m
Green	Y	X	-	-	Z	Y	e
Green	-	Z	Y	Y	X	-	m
Primary Basis	Y	X	X	Z	Z	Y	e
Dual Basis	Z	Z	Y	Y	X	X	m

Table 1: Here we define the Pauli frame for both anyon types by looking at the logical operators. The Primary Basis row has the same Pauli progression as the logical operator in Figure 9. Using this Pauli to anyon association, we can look at all of the detectors during the six ISG cycle. Looking at each vertical column shows that all detectors and logical operators are one of only two Pauli types during any given ISG. They naturally split into the two anyon types as defined by the logical operators, one anyon detector per plaquette color.

them to determine the Pauli frames of our anyons throughout the ISGs. Table 1 shows this Pauli to anyon association when defining the anyonic Pauli type by the associated logical operators throughout the ISGs. Primary vs Dual basis is arbitrary, but they are created from the opposite logical operators found in the code. Knowing the Pauli frame for the different anyons allows us to analyze the detectors from an anyonic lens. It turns out that all of the detectors break into one anyon type or the other. Every row in the table explicitly shows each detector's Pauli eveolution and associates it with an anyon type. Notice how each color of plaquette switches between being an e-detector and an m-detector. This is also an example of why the overall ISG period is six rounds long. Despite the Pauli frame appearing the same between the two detectors of the same color plaquette, they represent opposite anyon types.

With this, we can start to see why the Floquet code is actually just a surface code in disguise, catching errors in the same way the VSC does. Figure 10 shows a grouping of three plaquettes surrounding a single data qubit in the center. As

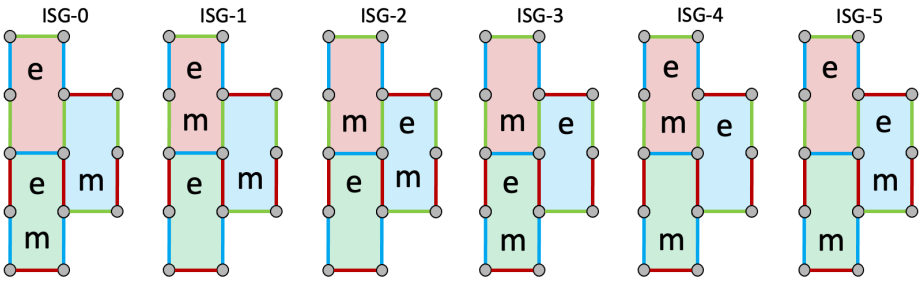


Figure 10: This image shows where the e- and m-type detectors live in the plaquettes surrounding a single data qubit. In every ISG there are exactly two detectors that will detect an error on the center data qubit. That means an error on the center data qubit will cause anyonic pairs as seen previously. This means we can use the same anyonic intuition for error strings, and use matching for decoding.

with all data qubits defined in the hexagonal layout, each of the three plaquettes are different colors. Using the information gleaned in Table 1, we can label where e and m anyon-type detectors are active for each ISG. It turns out that the data qubit is adjacent to two e-detectors and two m-detectors in every ISG. This is identical to what was seen with the VSC. Not only is this comparison nice, but it also means that the same matching decoding strategy will hold for the Floquet code.

4.4 Adjusting Measurement schedule to a CSS-style code

The original Hastings Haah definition of a Floquet code used orientation to define Pauli type [HH21], while here we associated Pauli type with color as is done in [GNM22]. Another possibility is to associate Pauli type with anyon type in a CSS-style, as is done in the VSC and [Kes+22] for their variation of a Floquet code. To achieve a Pauli anyon association, the color ordering of the measurement rounds does not have to change, but the Pauli type associated with each round gets altered. The main difference between the [HH21] and [GNM22] approaches and the approach in [Kes+22] is that in [Kes+22] the anyon types do

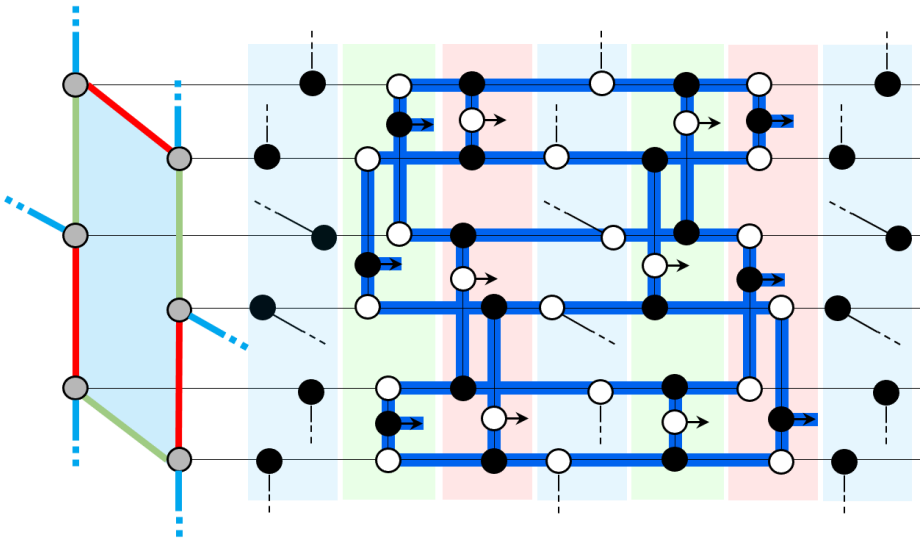


Figure 11: Above is the same detector that was shown earlier, but with the CSS-style measurement scheduling. Notice the new measurement scheduling leaves the Pauli type of the detector unchanged. A result of this is that we do not need to include the measurements that were used to change Pauli type in the previous case.

not change Pauli association as the code progresses through the ISGs. Looking closer at the Pauli progressions in the previous section, it can be seen that every measurement round alternates measuring the Pauli-type of the two anyons.

We can choose to set the primary anyon to a Z-Pauli frame, and the dual anyon to a X-Pauli frame. Then, we need to check that the detectors and logical operators still function correctly. Figures 11 and 12 show the CSS-style of the same detector and logical operator that were pictured previously in the Pauli = Color approach. The only difference here is that when a measurement round is of the opposite Pauli type, the results are not multiplied into the oberv-ables. This is possible because the check operators being measured commute with all observables of the opposite anyon type. This means that half of the measurements included in the previous detectors and logical operators were only included to update an anyonic Pauli frame, which is no longer necessary.

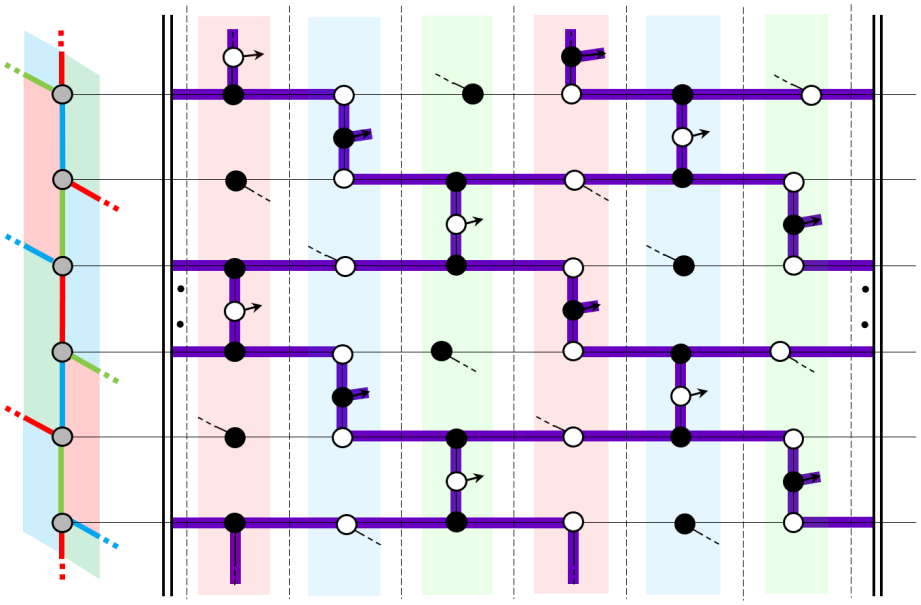


Figure 12: Above is the same logical operator that was shown earlier, but with the CSS-style measurement scheduling. The changes are the same as in the detector case. Here the Pauli type of the logical operator remains the same through its lifetime, and the measurements that were used to change Pauli type in the previous case now do not need to be included.

This new approach has the result of being much cleaner from an anyonic perspective, because it is much easier to keep track of the Pauli frames of all of the observables. The difference in the measurement cycles are compared below:

- P=C: Red-Z ; Blue-X ; Green-Y ; Red-Z ; Blue-X ; Green-Y
- CSS: Red-X ; Blue-Z ; Green-X ; Red-Z ; Blue-X ; Green-Z

Note that numeric simulations show negligible change in performance attributes such as threshold and lambda factor between all of the different anyonic Pauli frame approaches (P=C, CSS-style, and orientation). Later when we run the Floquet code on a real device, we will use the CSS-style approach.

5 Floquet Borders

So far we have defined the Floquet code on a torus, but to run this code on a real device it needs to be flattened into a planar version. This means it needs boundaries. The boundaries for the planar Floquet code defined here is going to be very similar to how the planar VSC was constructed. A bulk will be defined just like in the toric case again, then truncated plaquettes with be put on the sides to create the desired condensation conditions to store logical information.

Constructing these truncated plaquettes will be more difficult than in the VSC case. The VSC case was simpler because for a single truncated plaquette, the ancilla entangles only two data qubits instead of four before measurement. Because the plaquettes in the Floquet code are created from various link operators, we will have to introduce a truncated link to create a truncated plaquette. While referred to as a truncated link, really it's a direct measurement on a data qubit. Instead of a weight-2 parity check, we need a weight-1 parity check, which is just a direct single qubit Pauli measurement. Using direct measurements on data qubits must be handled very delicately to not damage the logical information encoded in the code.

To achieve this, truncated links are put on borders where normal links would be placed if the bulk continued. To ensure the correct condensation desired for a planar code, only one anyon type of truncated links and plaquettes are placed on each boundary. Figure 13 shows how this would look for a distance-8 5x5 code. The truncated links are shown by boundary data qubits being highlighted with a certain color. Light RGB colors for one anyon type, and dark RGB colors for the opposite anyon type. The abstract picture developed in the anyon section is also shown to the side to see how this code acts as the normal surface code. The truncated links are placed such that the planar code creates a 4-sided object, with opposite sides condensating the same anyon type. This construction will

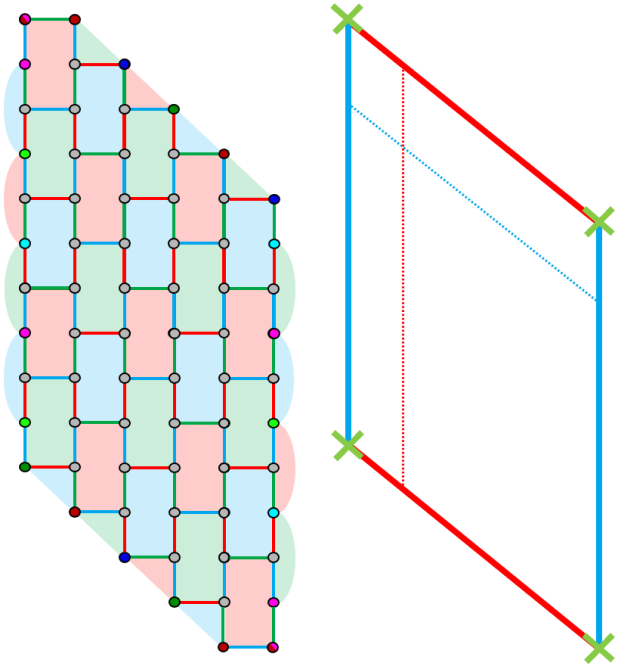


Figure 13: Above is an example of a planar layout of the Floquet code. The borders use truncated links to create truncated plaquettes of different anyon types depending on the side. The truncated links are shown by highlighting data qubits on the boundaries in a light or dark color associated with the link. The light and dark distinction is meant to show which anyon type the truncated link will measure. Truncated plaquettes on the sides will only be created for the anyon type shown by the truncated links associated with them. To show how this acts just as any other surface code from an anyonic perspective, our abstract representation of the code is shown to the right. Although it is now parallelogram shaped, this is exactly what was seen with the VSC.

ensure proper commuting for logical operator strings that are condensating on opposite boundaries.

Truncated plaquettes on the borders use two truncated links and two normal weight-2 links to create a detector. One such detector can be seen in Figure 14. This X-type detector starts by using a single weight-2 link and a single truncated link to create a three qubit detection region, half of the normal six qubit region. It goes through and commutes correctly with all of the rounds to then be col-

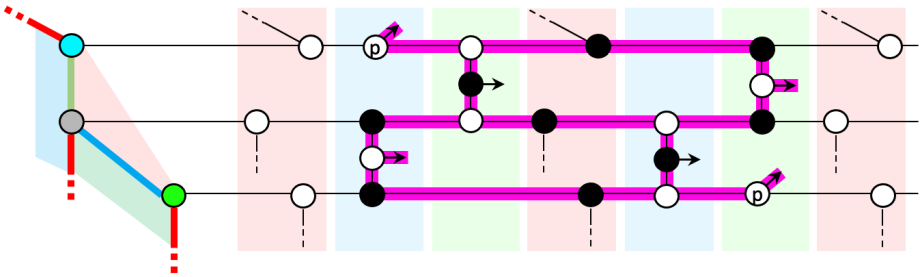


Figure 14: This Figure shows the life of a truncated plaquette using Pauli webs over a ZX diagram of the relevant qubits. The light truncated links will only perform a measurement when the correct anyon type is being measured, Pauli-X in this example. The nodes with a p in them represent projective measurements, which are handled specially with Pauli webs. How to handle the projective spiders is described in Appendix A.

lapsed in the green X-measurement round. Note there is nothing happening to the top and bottom qubit during the blue and green Z-measurement rounds. If a Z-type truncated link measurement would happen in those locations, it would scramble the detector, resulting in random results each time. This issue does not happen in the bulk because this qubit would be half of a full link operator that would commute with the full detector. Placing X- or Z-truncated links and plaquettes along the borders of the parallelogram shape then creates the desired borders to store logical information.

A complete view of a distance-4 2x2 code is shown in Figure 15, complete with both logical operators visualized on the ZX diagram with Pauli webs. While both operators are moving around the code, notice that there is always exactly one data qubit that is part of both logical operators. This is how we can see the two logical operators always hold the desired anti-commutation relations. In the toric case the logical operators would shift around the torus in one direction continuously. Here that also happens, but the truncated links on either end are used to take part of the logical operator into and out of the code.

The boundaries as defined here were selected because of their compatibility

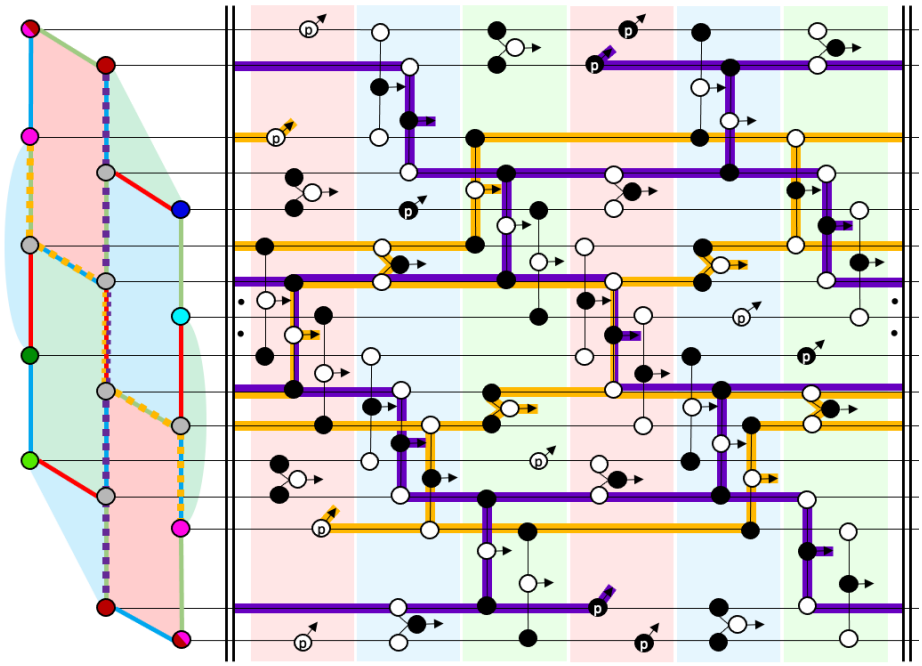


Figure 15: The smallest planar Floquet code given the construction defined here is shown above, complete with a ZX diagram of the full six ISG cycle. The Pauli webs for the two distance-4 logical operators are shown using Pauli webs overlaid. Notice how for every ISG, the Pauli webs overlap in exactly one location, ensuring the proper anti-commutation relations. The defined locations of the logical operators are shown with dotted lines on the layout to the left.

with IBM Eagle devices. The static nature of the boundaries make sure the code stays in place, and is more efficient in terms of footprint than other approaches. This style of direct measurement was defined in [Kes+22] and uses a different approach to constructing boundaries than in [HH22]. A similar approach was also used in [GNM22], but using bi-colored boundaries and weight-4 truncated plaquettes guided by [HH22] instead of the tri-colored boundaries with weight-3 truncated plaquettes used here. The difference in boundaries here given the constraints of the devices made the approach in [Kes+22] more appealing with the ability to fit multiple sizes on an IBM Eagle chip. Another boundary creation approach was also developed in [Pae+23], but it resulted in the code shifting

by a plaquette side to side to maintain correctly condensation boundaries. This extra motion made it the least desirable approach for a limited footprint device.

Another attractive variation was also to use a different color scheduling. Instead of sticking with the RBGRBG color scheduling, instead a RBGRGB approach is a common varient, which here will be referred to as the wiggle scheduling. With the standard scheduling, the logical operator needs to use the truncated link measurements to wrap the moving logical operator around the boundaries. This is not the most desirable case. The benefit of the wiggle scheduling is the logical operator just moves back and forth, never falling off the side of the planar surface. The reason it was not implemented here is because it requires bi-colored boundaries, which do not fit as well on the IBM chips as the tri-colored boundaries. Because the experiments being run here are simple $|0\rangle$ or $|+\rangle$ memory experiments, we decided to go forward with the boundaries described above for easier chip layout compatibility.

The simulation done in this work uses the open source Clifford simulator stim with the standard circuit level Pauli noise model as is common in the field [Gid21]. Decoding is done by Minimum Weight Perfect Matching (MWPM), implemented using pymatching [Hig21]. Figure 16 shows a threshold simulation for the Planar CSS-style implementation of the Floquet code, showing a threshold between 0.2 - 0.3%. This is consistent with existing benchmarkings of the Floquet code seen throughout the literature when tailoring simulation to a superconducting circuit based approach [GNM22; Kes+22]. This is lower than the VSC threshold which is agreed to be in the .07 - 1% range. A significant contributing factor to this is the size of the detectors in the Floquet code. Floquet detectors live through 5 measurement rounds, require a minimum of 6 measurements to be included in the result, and the plaquettes are of weight-6. In the VSC, detectors live for only 3 measurement rounds, only include 2

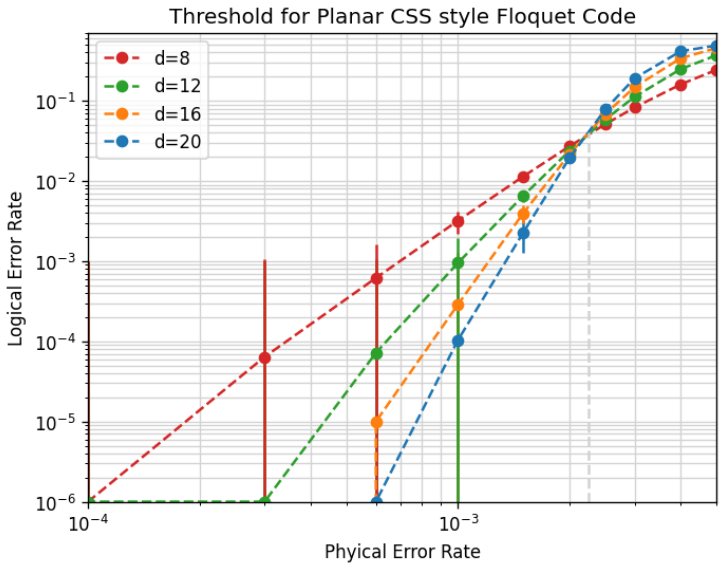


Figure 16: Threshold simulation for a planar CSS-style Floquet code. Each point is the logical error rate for a Monte-Carlo simulation, each with a given distance and physical error rate p. The number of sub-rounds run for each simulation was twice the distance, since each anyon type of detectors are measured every two sub-rounds. This ensures that the code is still robust to $(d-1)/2$ errors in the time dimension as well.

measurements, and use weight-4 plaquettes. One can think of the analogy of watching television in 480p vs 720p when comparing the sizes of detectors between the two codes, better resolution with smaller detectors results in a better understanding of what errors occurred.

6 Running the Floquet Code on a Real Device

The device used to produce the results shown in this section was IBM's Eagle device `ibm_sherbrooke`. `ibm_sherbrooke` was selected because it has the best noise rates of all of the IBM 127-qubit Eagle devices. The code that was run on the device was the CSS-style Planar Floquet code with the boundaries described in Section 5. Only two sizes were able to fit on the device, parallelograms of

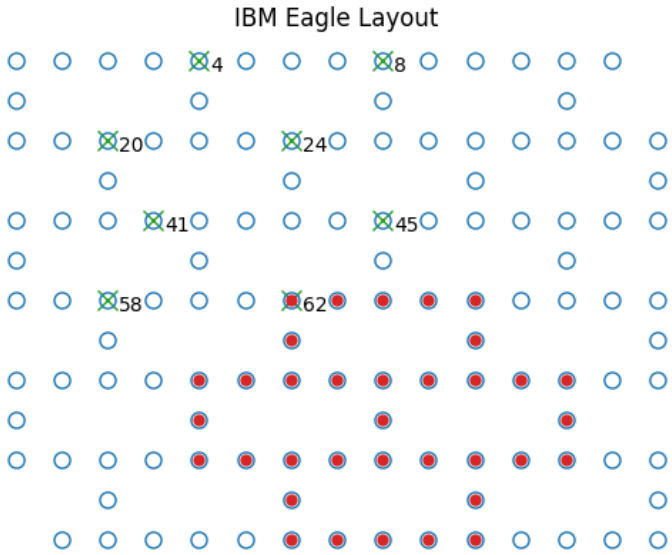


Figure 17: Shown above is the layout for an IBM Eagle quantum device containing 127 qubits. Every blue circle represents a qubit on the device. An example of how a 2x2 Floquet code could fit on such a device is shown in red. In total there are eight locations to place a 2x2 code, all are marked with green Xs with the index of the qubit next to it. There are only two locations to fit a 3x3, 4 and 24.

two or three plaquettes wide. The 2x2 acts as a distance 4 code, and the 3x3 acts as a distance 5 or 6 code depending on the ISG. Although only these two sizes were able to fit on the device, there were many locations that were able to run these codes. Figure 17 shows all of the different locations a 2x2 Floquet code could fit, where the index referenced corresponds to the qubit index of the upper left most qubit of the Floquet code. Meanwhile, a 3x3 Floquet code could only be placed at qubits 4 and 24.

First, we would like to determine the logical error rate per sub-round for the two sizes available. The data in Figure 18 shows the lifetime of both sized logical qubits in the $|0\rangle$ and $|+\rangle$ states. Only the best combination of location and dynamical decoupling scheme for each size is shown. The data is fitted with

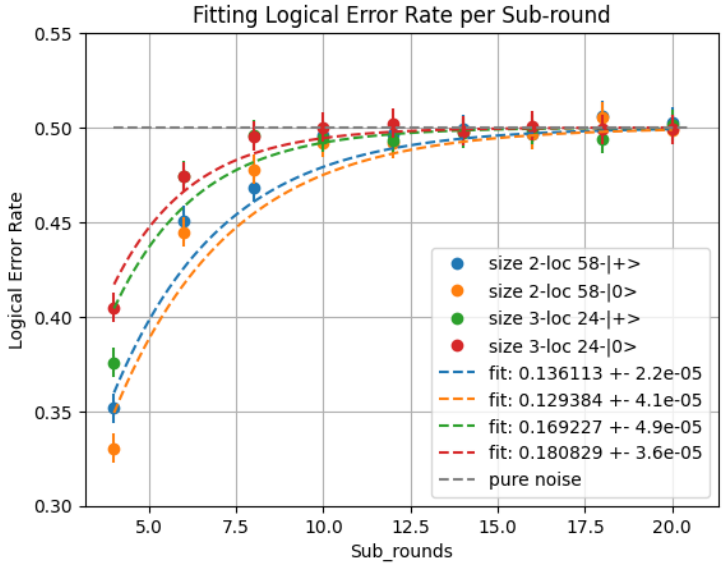


Figure 18: Two different sized codes run in both bases for a varying number of sub-rounds. A fit was then added to the data to obtain the logical error rate per sub-round. Notice that the 3x3 code performs worse than the 2x2, decaying to coin flips much faster. This is an effect of the device operating at noise levels above threshold.

$P_{error}(r_{sub-round}) = [1-(1-2\epsilon_{logical})^{r_{sub-round}}]/2$ to find $\epsilon_{logical}$, as is done in [Che+21], where $\epsilon_{logical}$ is the logical error rate per sub-round. It is clear to see that the smaller size code performed better than the larger one, indicating that the device is operating at noise levels above threshold. How far above threshold is difficult to characterize, and attempts to characterize this will be reviewed in detail in the next subsection.

To determine the best location and dynamical decoupling schemes, all variants were run on `ibm_sherbrooke`. The results of these runs can be seen in Figures 19 & 20. Dynamical decoupling is a strategy used to rotate out slow acting noise sources. Consider a slow noise source that causes the Bloch vector of a qubit to precess around the Z-axis. By applying an X-Pauli gate, the precessing is effectively reversed. By timing the locations of a pair of X-gates to

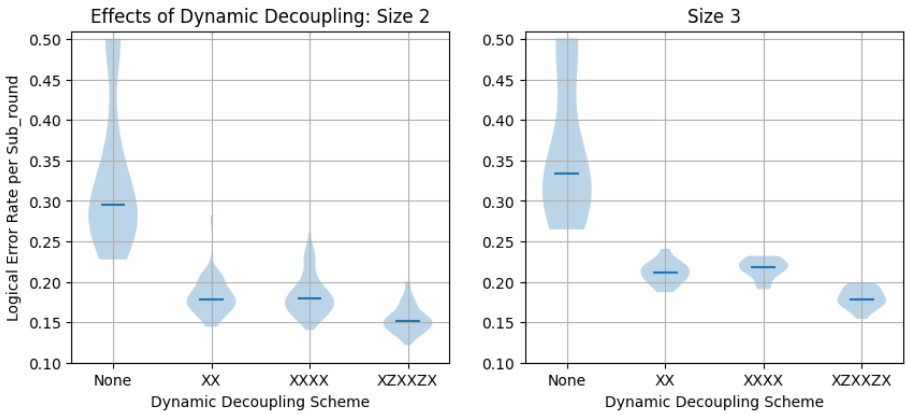


Figure 19: The data included in these violin plots include all possible locations, bases, and dates for jobs run on separate days. For every data set, a fit was calculated like in the previous image, and then they were grouped together by dynamical decoupling scheme. It is quite clear adding any dynamical decoupling has a large effect in maintaining a logical state, but the XZXXZX scheme definitely outperforms the other two tested.

make the time spent precessing in opposite directions equal, the noise cancels itself out. This type of noise cancelling is why $T_2^{(echo)}$ times are longer than standard T_2 times. The dynamical decoupling gate sequence is put in any gap where a data qubit is idling while other operations are done elsewhere in the circuit. The data in Figure 19 shows the effect of three different dynamical decoupling schemes compared to no dynamical decoupling. It can be seen that the XZXXZX scheme works the best, likely because it includes both Z- and X-gates to rotate out noise in multiple directions. Comparing all schemes to no dynamical decoupling, it is clear that dynamical decoupling is an essential tool to be able to run stable quantum circuits. Circuits were also run at every available location to find the best one. Each location uses a different set of physical qubits, each with their own noise profile. The comparisons between all locations are shown in Figure 20. Data for the violins were taken from running in different bases with runs spread out over separate days. Because of its consistent and

low error rate, location 58 was used in Figure 18.

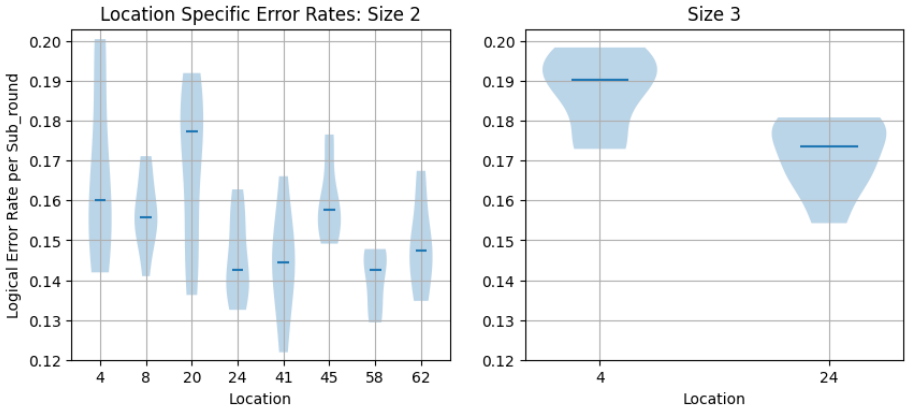


Figure 20: The data included in these violin plots use only the XZXXZX dynamical decoupling scheme, but include data from both bases and jobs run on several different days. We consider 58 to be the best locations for a 2x2 code. Not only does it have one of the lowest medians, but it is the most consistently low of all the violins. Clearly for the 3x3s, location 24 appears to be more stable.

6.1 Using Detector Likelihood for Benchmarking Real Devices

One issue when it comes to running quantum error correction circuits on real devices is that they are very difficult to simulate. The standard circuit level Pauli noise model sets all noise parameters equal to each other, which is not the case for real devices. Real life noise is made up of big and horrible Kraus operators with no nice structure in general. Using Pauli noise is a good approximation to see the overall dynamics of a code because the stabilizer measurements of a code project single qubit noise down to Pauli noise and it is simple and fast to simulate, but it is not accurate when trying to simulate a specific device. This discrepancy is well known, being characterized in [GHU23]. This has pushed experimentalists to develop more complicated and slower simulation techniques, which can still fall short in accurately characterizing quantum error correcting

codes run on real devices [Ach+22; Kri+22]. A good discussion about comparing simulation techniques can be found in the appendix of [Ach+22]. Although this is the case, much of theoretic QEC research uses this noise model for its convenience and speed, simulations done in this work included. As an example of the simulation to real device descripency, Figure 21 shows a heatmap of detector triggering likelihood for real device data on the left, compared to a simulation using device calibration data on the right. There is a clear difference in magnitude and location of noise between the two data sets.

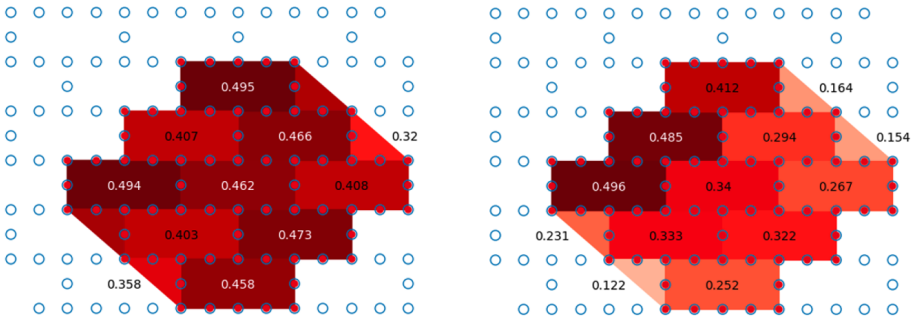


Figure 21: Heatmaps of Detector likelihoods. On the left is real device data, while on the right is simulated data with the device's calibration data. This image is meant to illustrate that simulations done even with qubit level specific noise models are still quite far removed from actually running these codes on the devices. Not only is the order of magnitude of the noise a lot higher for real device data, but the noise distribution spatially also does not line up well.

To analyze this discrepancy, we look at the average detector likelihood as an intermediate parameter to compare real device data to simulation data. This is in part due to the foundations of topological QEC codes. In the introduction when Kiteav first developed the topic, he drew a comparison between detectors and to spin flips in solid state systems [Kit03]. Comparisons between the Ising model from solid state mechanics and error correcting codes also extends to compare error correcting thresholds to the so-called Nishimori line, which defines the border between short-ranged and long-ranged orderedness of a system

[Che+23; Lee+22; Nis81]. Considering this model, the frequency of detector flips, analogous to spin flips in the Ising model, was worth exploring further as a characterization tool.

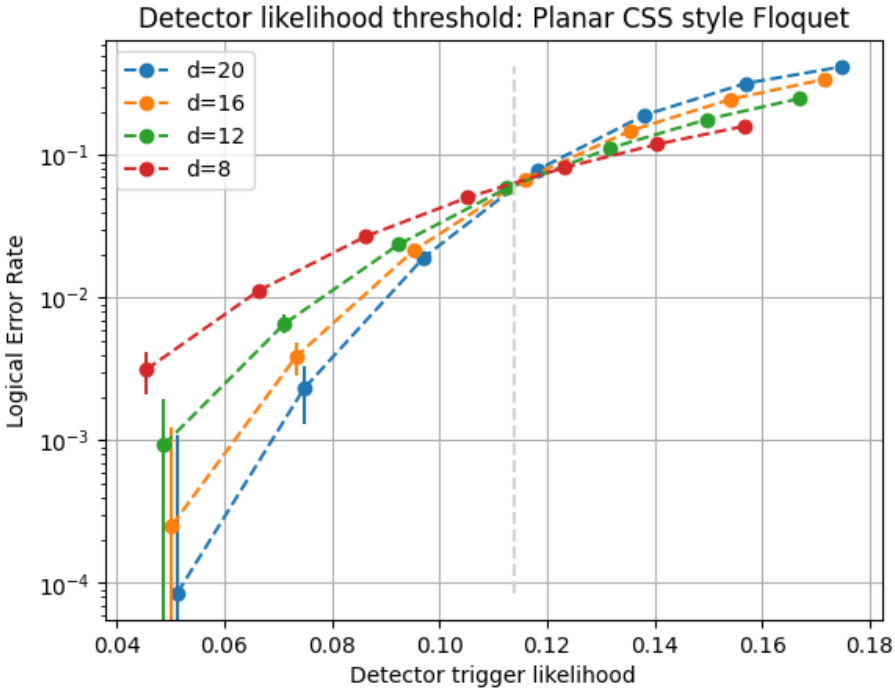


Figure 22: A threshold graph that instead of using a physical error rate p for the x-axis, uses the average detector likelihood. It can clearly be seen that different distances all converge like in normal threshold plots. We will see later that there is a very clearly defined relationship between detector likelihood and physical error rate p.

To start exploring this, we looked at simulation results to see if a detector likelihood threshold could also be seen. Figure 22 shows exactly that, with different distance Floquet codes intersecting at the same average detector likelihood, just like in normal threshold graphs. The simulation data presented in Figure 22 was run with a flat noise model. This means that for the standard circuit Pauli noise model, readout, initialization, two-qubit gate, single-qubit gate, and idle error probabilities were all set equal to each other. When it comes to

real devices, this is not the case.

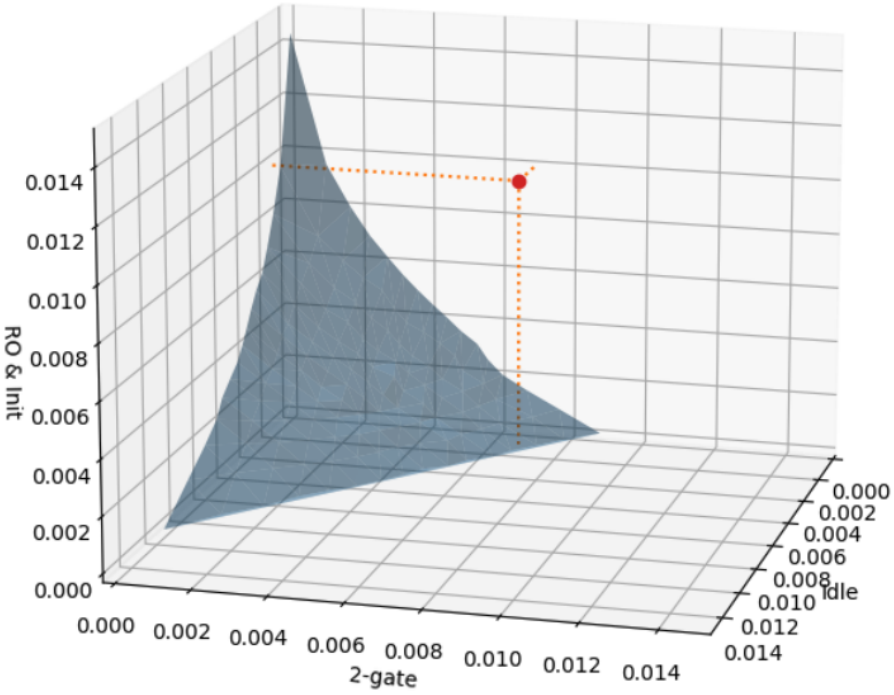


Figure 23: A threshold surface takes different noise breakdowns for the 3D noise space defined, and calculates a threshold for each direction. The threshold then acts as the radial magnitude for every point in the surface. This shows how different noise breakdowns affect the threshold of the code. The red point marked outside of the threshold surface represents the median noise rates for IBM's `ibm_sherbrooke` device.

To investigate different noise breakdowns we looked at threshold simulations throughout a 3-dimensional noise space. Figure 23 shows a threshold surface. Similar to those shown in [HW23], this threshold surface breaks the noise contributions into three main categories: idle noise, two-qubit gate noise, and readout and initialization noise. Readout and initialization noise are taken to be the same because initialization is done by active reset, where the main contribution to errors is the readout error since gate errors have a much higher fidelity. Single-gate errors were just pinned to a low value to maintain only three

degrees of freedom, and do not have a large effect due to their higher fidelity. A threshold is calculated for different directions in this 3D noise space, then all are plotted on the same surface. The specific shape of these surfaces are specific for each error correcting code, as shown in [HW23]. For the CSS-style Planar implementation used to create this surface, the surface takes on a concave pattern. Also marked in the image is where the `ibm_sherbrooke` median noise lives in this space, clearly outside the surface as we have seen the device is above threshold.

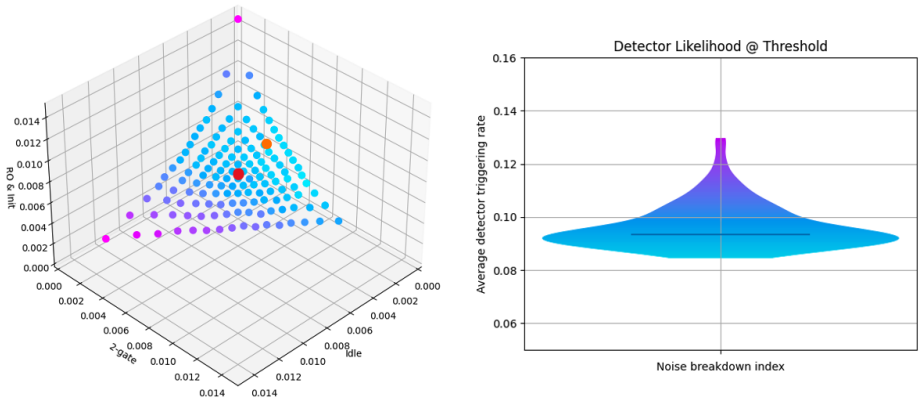


Figure 24: The two figures above are meant to convey the variance of detector likelihood at threshold throughout this noise space. To the left, the same threshold surface from Figure 23 is shown with a coloring scheme corresponding to the detector likelihood found at the threshold of the given noise breakdown. The top and the left corners can be seen as hot spots, where the detector likelihood threshold is significantly higher than in the rest of the surface. The red and orange dots in the center of the surface represent a flat noise model and the median `ibm_sherbrooke` noises' locations on the surface respectively. The right violin plot shown the values of the detector likelihood found throughout the surface.

Then, for every point simulated at the threshold surface, the detector likelihood was also calculated. This will inform if the detector likelihood threshold is the same over the entire threshold surface. It turns out that is not the case, and the detector likelihood at threshold varies for different noise breakdowns.

Figure 24 shows this distribution. There is a distribution of detector likelihood thresholds throughout the threshold surface, where it increases toward the extremities in the corners. The locations of a flat noise model and the median `ibm_sherbrooke` noise breakdown are also marked on the threshold surface to show where they exist in this space. Luckily, despite the difference in detector likelihood between the two noise breakdowns, the flat and `ibm_sherbrooke` noise breakdowns can be seen to have roughly the same detector likelihood threshold, both living in the bulge around the median value in the right plot. The hope is that the difference between the two is small enough that using a flat noise model to simulate a real device gives an accurate characterization of the device.

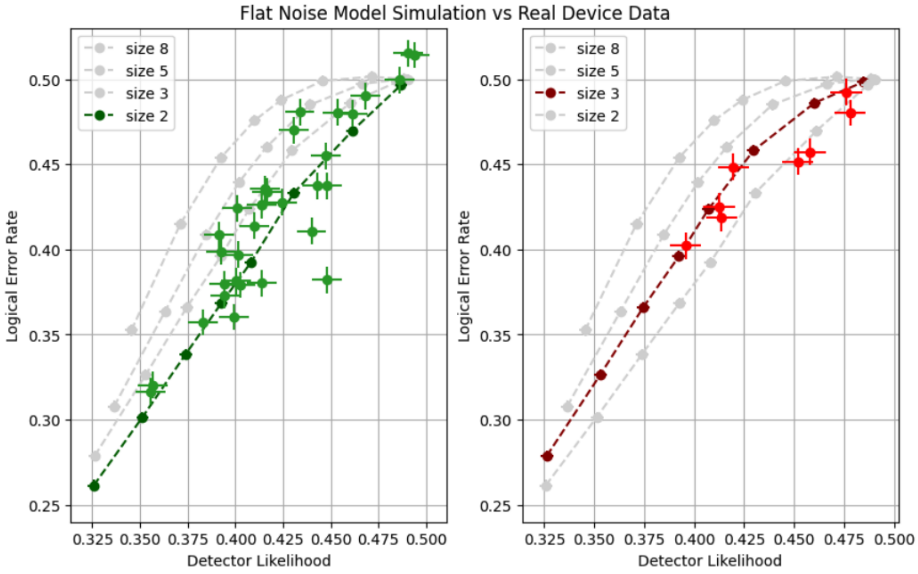


Figure 25: This figure attempts to show the parallel between simulation data and real device data by the use of the mediating parameter of average detector likelihood. Data for 2x2 codes is shown on the left in green, while data for 3x3 codes is shown on the right in red. The dark points connected by dashed lines show simulation data with a flat noise profile of a physical error rate p. The light points that have larger error bars are runs on a real device. Different runs represent different locations and dynamical decoupling schemes run for four sub-rounds because the lowest number of sub-rounds has the clearest signal. Different simulation data for other distances are also shown in the background in grey.

Now to attempt comparing real device data to simulated data using the average detector likelihood metric as an intermediary. Figure 25 shows this comparison. On the x-axis is the average detector likelihood, while logical error rate is plotted on the y-axis. These are the same axes as the threshold graph seen in Figure 22, but zoomed in much closer to the (0.5, 0.5) point of pure noise because the device is not close to threshold. The darker dashed lines show simulation data with a flat noise model, while the lighter points overlaid show data taken from `ibm_sherbrooke` for various locations and dynamical decoupling schemes. The variation in location and dynamical decoupling gives a natural variation of noise parameters coming from our device for more diverse data. Although the matching is not perfect, it can be seen the two data sets roughly trend together for logical error rate as a function of detector likelihood.

So far this correspondence shows promise, despite the variation with noise breakdown. To try and take advantage of this correspondence, lets define a new metric called effective-p. Effective-p represents an associated flat noise model value that can be associated with a device. It is easy to find this effective-p value given an average detector likelihood extracted from a circuit run on a real device. By simulating the detector likelihood for various p values, then fitting it with an exponential saturation function, an effective-p rate can be calculated from a device's average detector likelihood using the simulated fits. Figure 26 shows these fits for varying sized codes for 6 sub-rounds, or a full ISG cycle. Different sizes and sub-rounds need to be fitted because the ratio of boundary detectors vs bulk detectors vary with the dimensions of a code, which plays a role because the boundary detectors are half the size of the bulk detectors. This boundary to bulk ratio slightly affects the fitting as a result.

To see how well an effective-p simulation matches with real data, simulations were done to compare three different noise models' accuracy when simulating

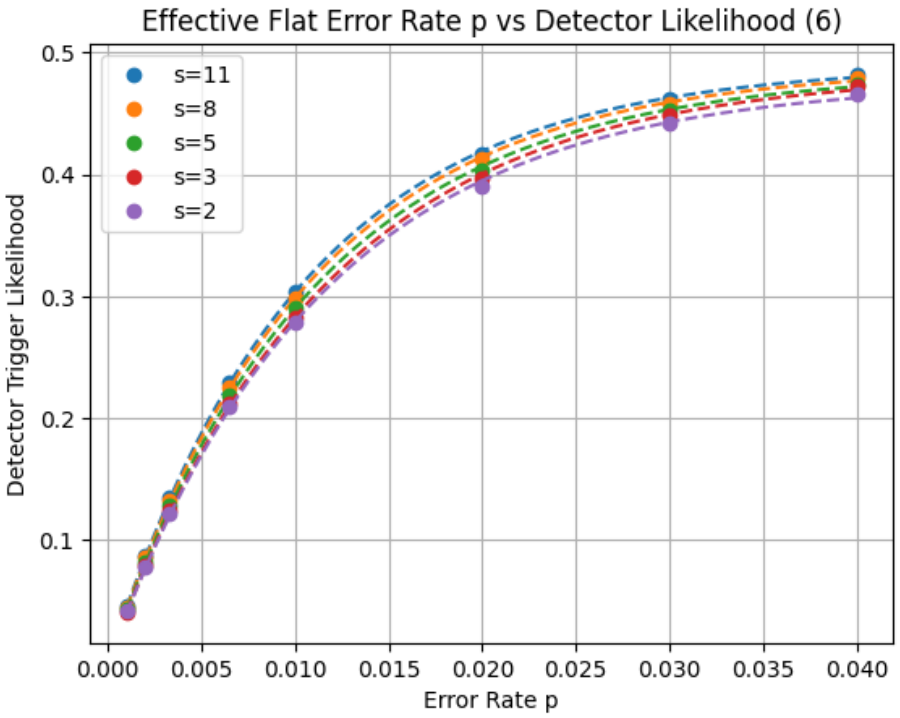


Figure 26: The points here represent simulation data at a given physical error rate p to calculate the associated average detector likelihoods. The dashed lines are exponential saturation fits, whose data is stored for later effective-p fitting.

the experiment who's real device data is seen in Figure 18. The three different noise models simulated include median noise, calibration noise, and effective-p noise. Median noise takes the median noise values for a device and sets all parameters in the simulation as such. For `ibm_sherbrooke`, these values are shown in Table 2. Calibration noise takes the calibration data from the device, so every source of noise is specific to every qubit or connection. Effective-p used the average detector likelihood averaged over the longest 3 runs of the real device data shown on the same plot, and used the relationship derived in Figure 26 to calculate an effective-p value. That value was then used in a flat noise simulation where readout, initialization, single- and two-qubit gate were all set equal to the

Noise Source	Median
Readout & Init	1e-2
Two-qubit Gate	7e-3
Single-qubit Gate	1e-4
Idle	2e-3

Table 2: Median error rates for `ibm_sherbrooke`. Initialization is done with active reset, and because the gate fidelity is so high, we set readout and initialization noise equal to each other. Idle noise is calculated from the worst of T_1 and T_2 times to err on the side of more noise.

extracted effective-p value. The results are shown in Figure 27. This effective-p method creates a much better fit to data than the other mentioned simulation strategies. While it might be slightly approximate, it does give a convenient single value to look at when benchmarking real devices. The effective-p value of a chip can then be firmly be put it into a broader simulation context. It also can be used to characterize the overall noise of a chip, which generally is a very complex landscape of different noise sources with variances spanning over the chip. Having a single value to compare the overall performance of different devices would be a useful thing to have.

All different combinations of data run on a real device of location and dynamical decoupling were also analyzed using this effective-p value to see if the improvement seen in Figure 27 holds for all data obtained. The results are shown in Figure 28, comparing the logical error rate per round of the actual device data vs the effective-p simulations. The data here was post selected to only show runs with detector likelihoods under 45%. This post selection is justified because our fitting model seen in Figure 26 becomes quite unreliable at very high noise rates, and the signal to noise is already very low to begin with. We care less about fitting coin flips accurately and more about simulating actual logical signals. It can be seen the points roughly follow the dashed line that represents perfect simulation. Looking at simulation accuracy for the point

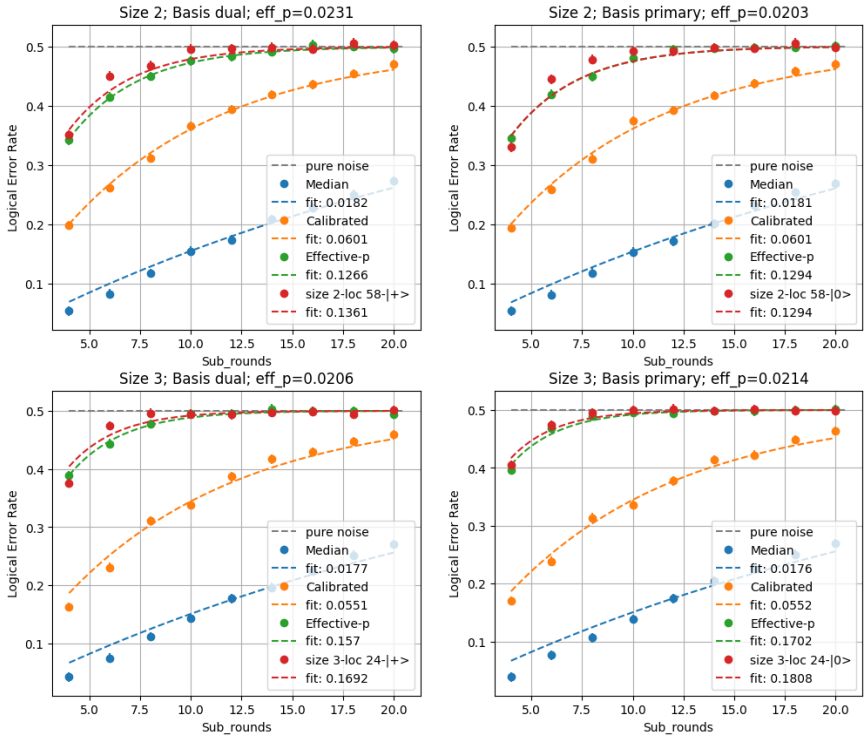


Figure 27: Here we take a look at the best location and dynamical decoupling schemes for each size that fits on an IBM Eagle chip. The red lines show the same data from Figure 18. Each is split into a different sub-graph so that three different simulation strategies could be compared with the real device data. Blue shows using median noise rates for the five different noise categories, orange shows using the device calibration data, and green shows a simulation with the derived effective-p value.

marked with the red arrow, which represents the worst accuracy of the 19 runs shown, it still yields a much better fit that other methods as can be seen on the left-hand side of the figure. Even for the worst case encountered, the effective-p simulation still greatly outperforms the compared simulation techniques.

Two very big caveats should be brought up with this data fitting, both have to do with this device being very noisy. First is that, as can be seen in Figures 18 & 27, there is barely any signal with current devices for Floquet codes. Second is that this is all above threshold. We would expect this agreement to only get

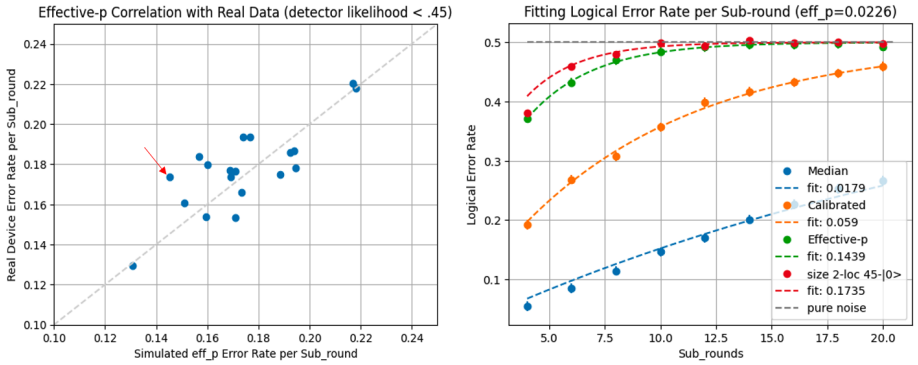


Figure 28: The right image shows real device $\epsilon_{sub-round}$ vs simulated eff_p $\epsilon_{sub-round}$ for all possible combinations of location, dynamical decoupling, and basis post-selected for detector likelihoods of <45%. The worst agreement of all the 19 post-selected runs is marked with a red arrow. The left-hand image shows the comparison of the fit of the worst agreement, which still outperforms other simulation techniques.

better with lower error rates, but nothing except intuition says that. If we go back to the Ising model analogy of topological codes, we are in a completely different phase of matter than we would like to be. The experiments here are like trying to figure out how to characterize ferromagnetic magnets when all you have is a normal piece of paramagnetic metal.

7 Conclusion

In this work we used the anyonic perspective of quantum computing and ZX calculus as analytical tools to understand the Floquet code. We then selected borders that fit best with existing IBM quantum devices to run these codes on real devices. Ultimately, running the codes showed that current devices perform at noise rates well above the Floquet threshold. Given these results, the focus of the project pivoted to using the Floquet code as a benchmarking mechanism. The average detector likelihood of a code was defined as an intermediary parameter to calculate an effective flat Pauli noise model error rate

we called effective-p. This effective-p was then shown to result in much better simulation data using a standard circuit level Pauli noise model that other techniques. The effective-p value, while not exact and maybe less accurate for different noise breakdowns, gives a single value to reference that incorporates the total noise performance of a system. This could be very useful for comparing different devices and placing a real device into the context of simulation results more accurately than previous techniques. Using this benchmarking technique, it was seen that `ibm_sherbrooke` shows an effective-p of just over 2% for the best locations on the device.

Appendix

This appendix has five sections, but it really breaks down in to two main parts.

- Appendix A gives a review of ZX calculus as it applies to Quantum Error Correction to understand the use of ZX diagrams and Pauli webs in the main text. This is a proper length section in its own right.
- The other sections form the second main part of the appendix and are quite short. They consist of a couple of images of interesting plots and visualizations that were not included in the main text.
  - Appendix B shows an example of the matching graph for the Floquet code.
  - Appendix C shows real device vs simulation data like in Figure 25 of the main text for a different number of sub-rounds.
  - Appendix D shows a blank ZX diagram of a 2x2 Floquet code so a reader could try drawing their own Pauli webs.
  - Appendix E shows what we call a weather report circuit. The weather report circuit was a Floquet code without borders fitting the entire device in attempts to find the best location to run a Floquet code. When running a circuit of that size, the noise takes over and we got no useful results out of it, but a brief discussion is worth including.

A ZX Calculus for Quantum Error Correction

This appendix section is intended to give an overview of ZX calculus basics as it applies to Quantum Error Correction. This is by no means a comprehensive review of ZX calculus, but instead to be used as a quick guide to understand the ZX diagrams and Pauli webs illustrated in the main text. There are many good resources in relation to ZX Calculus [Coe23; BPW17; Bac+21; Dun+20; Kis22; Wet20] and Pauli Webs [Bom+23b] that may be useful to the reader for a more in depth introduction to the topic.

A.1 ZX Basics

ZX Calculus is a graph based formalism used to represent quantum states and processes. It is easy to convert any quantum circuit into a ZX diagram and apply various transformation rules. These transformation rules give a mathematically complete set of tools to find equivalent circuits [Wan23]. This outline will not go into more powerful transformations, but instead review what is necessary for the applications in this work. A good review paper for learning the complete set of rules for ZX Calculus can be found in [Wet20].

A.1.1 Spiders

The most basic building block of ZX Calculus are Spiders. Spiders are the nodes of a ZX graph, and come in two different colors. Black nodes represent X-spiders and white nodes represent Z-spiders. Spiders may also come with a phase, which is always written either in the spider, or directly next to it. Luckily, most fault-tolerant circuits can be expressed with phase-less spiders. When a spider does not have a phase it is omitted. Figure 29 has examples of general spiders shown without a phase, as well as the other features described in the rest of this subsection.

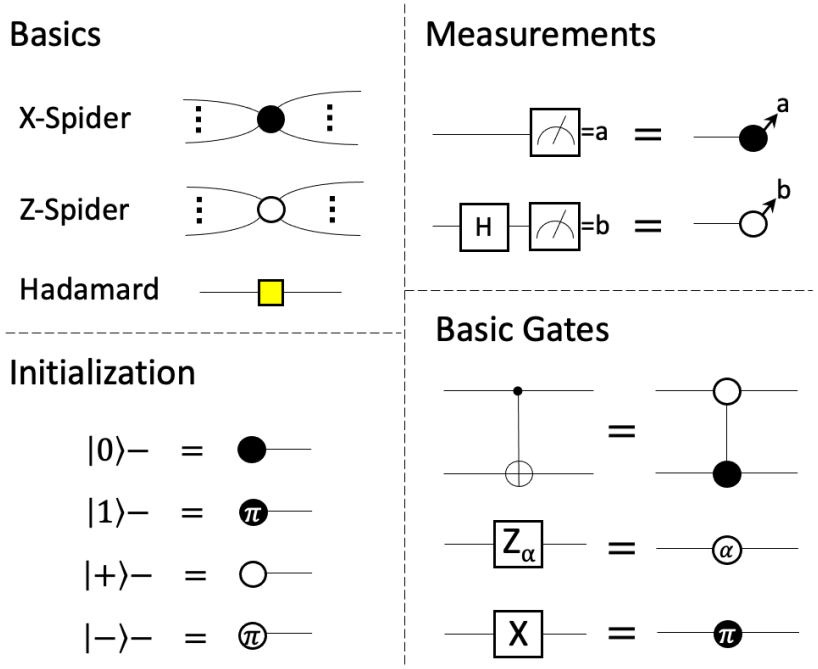


Figure 29: Here are the basic ingredients for transforming a quantum circuit into ZX diagrams. The most basic unit is the spider, coming in both X- and Z-Pauli types. Spiders are connected by edges (or ports by some nomenclature) to other spiders or Hadamards. The transformations for measurements, initialization, and some basic gates from quantum circuit language into a ZX graph are all shown above.

Edges are used to connect different spiders in the graph. Edges typically represent the identity channel acting on a specific qubit, or connections between two qubits during multi-qubit gates like CNOTs. Sometimes an edge with have a yellow box on it, which symbolizes a Hadamard gate. In the notation used in this thesis, there may also be an arrow coming out of a spider. This symbolizes the classical output of a quantum device performing a measurement.

One difference between the spiders pictured here versus other ZX Calculus literature is the color scheme. Normally, the Z(X)-spiders shown in white(black) are pictured in green (red). Using black and white is a common color variation of ZX diagrams. This variation make it easier to draw ZX graphs with a single color

and is less constrained by specific color associations, which will be beneficial when drawing Pauli webs

A very common use for ZX Calculus is to transform a quantum circuit into a ZX diagram, and use the transformation rules to manipulate the circuit. Fig.29 shows how to convert qubit initialization, measurement, and some basic gates from a circuit representation into the ZX language. An example of using these rules to create a ZX diagram from a quantum circuit can be seen in Figure 31 while deriving a ZX representation of a parity check.

A.1.2 Transformation Rules

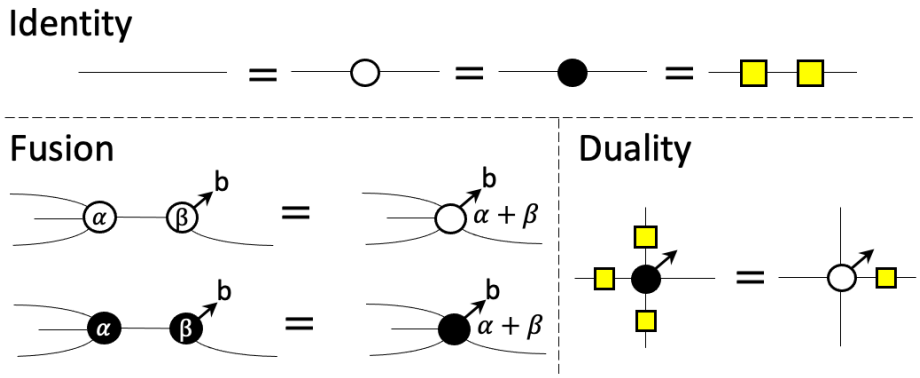


Figure 30: After converting a quantum circuit into a ZX diagram, the power comes from the manipulations that can be performed by the transformation rules. Some of the basic rules that will help derive the circuits necessary for this work are shown above. These transformation rules can be used in both directions.

Transformation rules are used to manipulate ZX graphs such that the resulting diagram is equivalent to the starting diagram. This is very useful as a tool to manipulate circuits knowing that the result will always perform the same action on a system, and has been used in circuit optimization, circuit equivalency validation, and analysis of quantum error correcting codes and lattice surgery [Wet20]. Figure 30 shows the transformation rules useful to derive parity checks

which are heavily used in quantum error correcting circuits. More complicated transformation rules such as the bialgebra rule and the Hopf rule are not include here.

The identity equivalences are the easiest to see. Two Hadamards cancel each other out because the Hadamard is its own inverse. A Z-rotation by 0 degrees is just an identity channel, so we are able to remove it from the diagram. The same goes for a phase-less X-spider, as it also performs no action on a system.

The fusion rule can be used to combine or break up spiders of the same color. When this is done, the overall phase of the spiders must remain the same. This is useful for collapsing a ZX diagram into its simplest form, which can be quite a bit more compact. Breaking out spiders can also be useful, and a good way to introduce ancilla qubits to help implement a certain operation.

Duality gives a tool to change the color of any node. Just like Hadamards can be used to alternate between X- and Z-bases in a circuit, they can be used in ZX calculus to alternate between X- and Z-spiders. This is done by placing Hadamards on every edge attached to a node, resulting in a flipping of the spiders color. By extension, placing Hadamards on every edge surrounding a full sub-graph can be used to flip every spider type withing the sub-graph.

Using these rules in reverse is also very useful as well. Being able to create more complicated circuits from a basic gate can be useful when native operations of a system do not support the gate. One such example of this is lattice surgery implementations of gates without easy transversal implementations like CNOTs and T gates [BH20].

A.1.3 Deriving Parity Checks

The rules above can be used to create simpler representations of parity measurements using ZX calculus. Figure 31 shows how to achieve such a the transformation for a ZZ-parity check. The standard way to measure a ZZ parity is

through the use of an ancilla qubit and CNOTs. The measurement of the ancilla at the end of this process projects the qubits into a particular parity subspace. By use of the fusion rule after converting the ZZ-check into a ZX diagram, it can be represented by a simple 3 spider diagram. Deriving the XX-check can be done similarly with the use of a duality argument, while the YY-check is a bit more complicated.

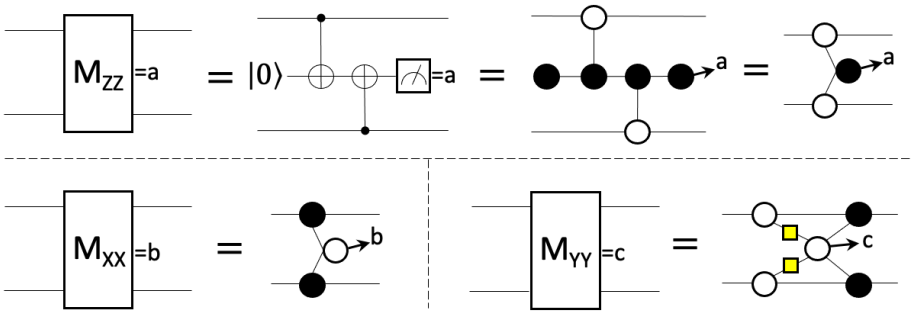


Figure 31: Parity checks will be used heavily in Floquet codes, so lets see how to simplify the ZX diagram of a parity check. The top row shows such a transformation. First the circuit with two CNOTs used to implement the parity check on an actual device is transformed into a ZX diagram, then the X-spiders where the ancilla qubit was are all fused together. This leads to a much simpler representation of a ZZ-parity check. The simplified representations for XX- and YY-parity checks are also included in the bottom row of the figure.

A.2 Pauli Webs

The concept of Pauli webs was first developed in [Bom+23b], with a similar concept being used in [MBG23]. Pauli webs act as an overlay on ZX diagrams to identify important aspects of quantum error correcting codes like detectors and logical operators. These webs will always start and end at either measurements or a qubit initializations.

Note the different sections of Figure 31 all use different colors in a light and dark scheme to express Pauli webs. This allows for having multiple webs on the

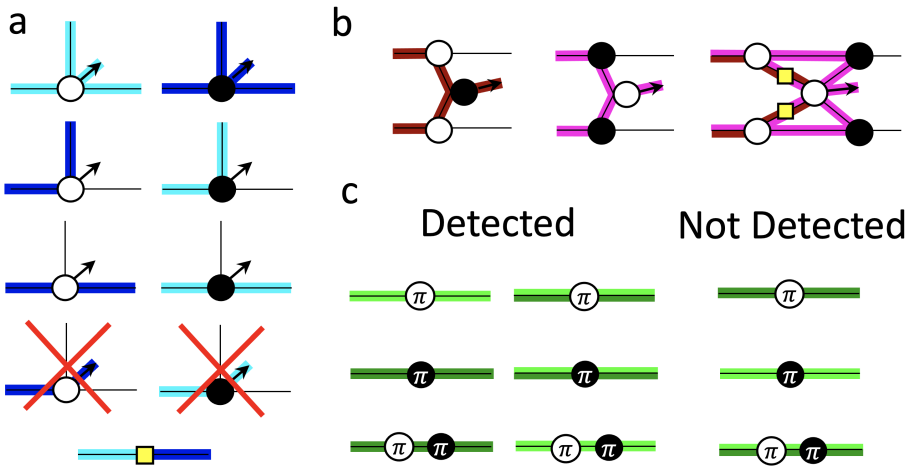


Figure 32: Here are the rules and intuition of how to draw Pauli webs. We define the difference of Pauli web by the shade of the web. Light webs represent X-Pauli webs while dark webs represent Z-Pauli webs. This gives flexibility in color choice of the web. Flexibility in color is very useful to show multiple Pauli webs on the same ZX diagram. The different sub-parts of this figure show this by having three different colors being used to show examples of Pauli webs. a) shows the rules of how to propagate different web types through different spider types. The general rule of thumb is that webs touching a spider of the same shade must propagate down every port attached to the spider. For webs and spiders of opposite shade, the overall parity must be even. To include the measurement outcome in a Pauli web, the web must be of the same shade as the spider associated with the measurement. If a Pauli web encounters a Hadamard, it much change Pauli type. b) Examples of how each type of parity check can catch the appropriate incoming Pauli webs. Notice that for the YY-check, the incoming webs show both shades, representing a Y-web. When both these colors encounter a spider, they each are able to separate, following their specific propagation rules. c) Examples of what types of errors are detected by Pauli webs. A pi-phase Z-spider (white) represents a Z-error, while a pi-phase X-spider represents a X error. Y-errors are shown with a combination of both, just like with the Pauli webs. Notice how a Pauli web cannot detect an error of the same type, because the error commutes with the web's detecting region.

same graph through the use of multiple colors. This is very useful when seeing how different detectors and/or logical operators interact with each other, while being able to tell the different Pauli webs apart.

When drawing Pauli webs, the different shades correspond to different Pauli

types. Unlike with spiders, light(dark) webs correspond to a X(Z)-Pauli type. The basic rule is that if a web flows into the same shade of spider, light into light or dark into dark, then it must be propagated down all edges associated with that spider. For a web going into an opposite shade spider, the number of incident webs just needs to be even. Another way to think about it is that every web going into a opposite shade spider needs an associated web going out. These relations are pictured in Figure 32. Note that to terminate a Pauli web at a measurement pictured with an arrow, the Pauli web must be of the same shade as the spider associated with the measurement. This disallows the two crossed out cases in Figure 32a. If a measurement is marked with a Pauli web, the result of that measurement must be included in the resulting value of whatever the Pauli web represents. Lastly, when a Pauli web encounters a Hadamard, the Pauli type much change.

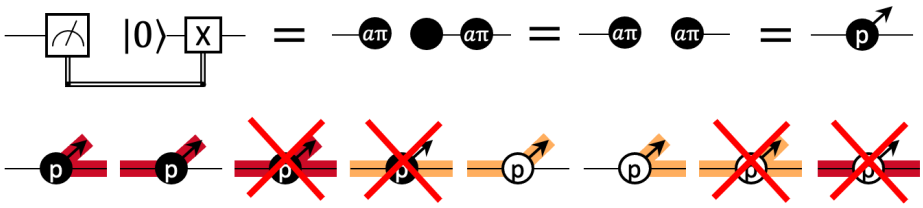


Figure 33: The circuits defined in the main text use direct measurements on data qubits are part of the measurement schedule. This is not well represented in the literature, so here we define a projective spider for convenience. The circuit to the left in the top row shows a circuit that is equivalent to a projective measurement in the Z-basis, collapsing the state to either $|0\rangle$ or $|1\rangle$. This is difficult to propagate a Pauli web through because technically there is a gap in the Pauli web due to the non-coherent nature of the operation. So we define the projective spider for a shortened notation. Opposite shade Pauli webs are not allowed to touch these projective spiders, because the measurement would be disruptive to their state. For webs of the same type, they are forced into the measurement arrow because the classical outcome needs to be included in the web.

Here we will also introduce one additional piece to the ZX lexicon. From the Author's perspective, in all of the ZX papers there is no convenient way to

show a projective measurement being done on a single qubit. Figure 33 shows a circuit that acts as a projective measurement, for convenient conversion into a ZX language. Ultimately before and after measurement, the ZX diagrams are completely separate from each otherexcept for a signle bit of information. We introduce a spider with the letter P in it, representing a projective measurement in a compact fashion. When thinking about how Pauli webs interact with this projective spider, opposite Pauli types are not allowed to interact with a spider. This is because the projection will completely scramble the web. For Pauli webs of the same color, regardless of which side the web comes in from it must be pushed to the measurement arrow. This is because technically there is a gap between the left and right port, so there is no connection between the two sides except the measurement outcome a, which needs to be included in the web. We will need this notation in the main text when discussing truncated link operators, which are direct measurements on data qubits. These Pauli web rules for projective spiders are shown in the bottom row of Figure 33.

A.3 Detectors

Now armed with a knowledge of how Pauli webs are produced, it is possible to find useful code characteristics with them. Looking at a distance 2 Vanilla Surface Code on the left in Figure 34, two separate detectors are shown. One in light blue, because it is a lighter color we associate that with X-type detectors (catching phase flips), and one in dark red associated with a Z type detector (catching bit flips). Two different colors are chosen because they represent two independent detectors. Focusing on the light blue X-detector, it is made up by two weight-2 XX parity checks on the border of the distance 2 code. The outcome of the detector is gained by multiplying the values of the two highlighted measurement outcomes modulo 2. If the detector is triggered that

means the Pauli web caught an error.

What makes this web slightly more complicated than in a repetition code is that it has to commute with the weight-4 Z-stabilizer measurement that happens in between the two XX-checks. The Pauli Web can be seen leaking into the weight-4 measurement due to the X-web interacting with Z-spiders, forcing the web to propagate into the weight-4 check. Luckily, they meet perfectly at the X-spider resulting in an even parity. This is a ZX visualization of a detector commuting with a stabilizer measurement of an orthogonal basis. The same can be seen with the larger dark red weight-4 detector commuting with the two smaller XX-checks. This representation gives a better intuition of time sensitive commuting properties. While with the VSC you can see the commutation relation between the X- and Z-stabilizers just from looking at a flat picture (shown to the left of the ZX diagram). In the Floquet code, the Instantaneous Stabilizer Group is changing all the time. Spreading out the visualization of a circuit into the time dimension allows a viewer to see time-sensitive commutation relations that are otherwise difficult to see with other visualizations. Additionally, drawing a detector that has commuting problems becomes obvious very quickly. Following the rules correctly will result in a spreading out of the Pauli web in an obviously unrecoverable manor, notifying the drawer that something is wrong with their intended detector.

The type of the web also determines what type of errors a detector detects. Figure 32c shows which Pauli webs catch which types of Pauli errors. A phase flip error is represented by a pi-phase Z-spider, a bit flip error as a pi-phase X-spider. A Y-error (up to a phase) is represented as a combination of a X-spider and a Z-spider, both of pi-phase. The errors shown on the left of Figure 32c are all able to be detected, while the 3 errors on the right are not able to be detected. It can be thought of as each error caught in a web of the same shade

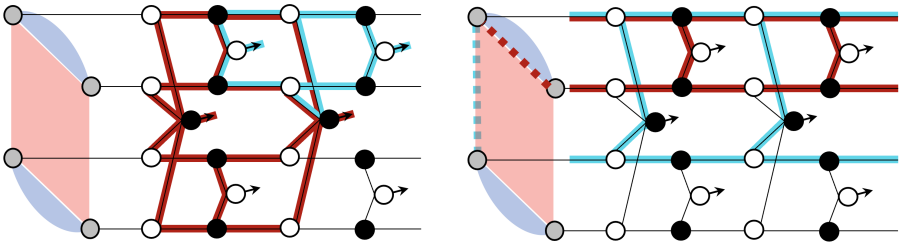


Figure 34: Here are ZX diagrams for a distance-2 VSC. Two measurement cycles are shown, each containing a weight-4 ZZZZ-check followed by two weight-2 XX-checks. The left figure shows two detectors, one X-type and one Z-type. The detectors are defined by taking the product of the same stabilizer in two successive rounds. A Pauli web is used to show the detection region of the each type of detector. Notice the Pauli webs leaking into the the opposite type detectors. This represents the ability to catch hook errors if they happen in the correct location. The "leaking" into detectors of the opposite type always happens with even parity, which is required for the two stabilizers to commute. If this wasn't the case, the Pauli web would not be contained.

(light or dark) flips the overall detector value. Since this happens twice for a Y-error in a two toned web, the overall parity is even. As a result, a Y-error is undetectable by a Y-web. This is desired because a Y-error should commute with a Y-detector, but be caught by the other two.

In Figure 35, all detectors are shown on the same ZX diagram for a distance-5 repetition code that includes initialization, three measurement cycles, and readout. The overall detector coverage of the code can be seen visually. Every edge has one or two dark colors associated with it, which means that if a bit flip error occurs on any edge, at least one detector will detect it. Edges associated with two qubits represent errors that are cause by two webs, while the boundary edges dangling on the top and bottom represent errors that occur in locations that are only covered by a single Pauli web. It is easy to see in this picture how the matching graph directly relates to its associated circuit.

Consider a bit flip error happening where the red X-is located. The bit flip would cause the triggering of the top dark blue and dark gold colored detectors.

Those detectors are associated with the same colored nodes in the matching graph, marked with red stars. The easiest edge to match is obviously the edge connecting the two nodes. That edge is associated with any location the where two detectors overlap on the same edge. Looking back at the ZX diagram, there is only one location in where the two Pauli Webs overlap. Having such a clear parallel between the matching graph and ZX diagrams with Pauli Webs give a powerful tool to analyze detector coverage of quantum error correcting circuits. Using this example, it is easy to see how the CNOT ordering of the ZZ-checks gives rise to the diagonal edges in the matching graph.

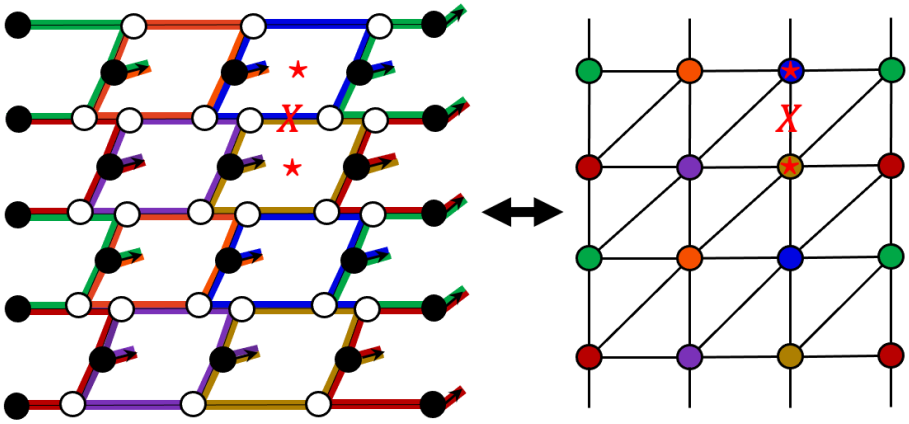


Figure 35: To the left is the ZX diagram for a distance-5 repetition code through a short lifetime of initialization, three measurement rounds, then readout. Every detector associated with this code is overlaid with different color Pauli webs. The right shows a matching graph used for decoding. The colors of the Pauli webs and the nodes of the matching graph help show how different errors trigger different pairs in the matching graph. The red X is an example of this. An X-error at that location would trigger two detectors, and there is a edge in the matching graph that corresponds exactly with a error in that location. Through matching, the edge tells exactly where the error occurred.

A.4 Logical Operators

Logical operators can also be viewed through Pauli Webs. The diagram on the right-hand side of Figure 34 shows how both logical operators live in the same distance-2 Vanilla Surface Code seen before. Here the logical operators are static, so they never move from the data qubits where they start. Seeing where the two logical operators intersect is also of great importance. The top qubit is where this intersection occurs and it can be seen that the logical operators will anti-commute through the entire circuit. With static logical operators this can also be seen just by looking at a picture of the layout, but in Floquet codes where the logical operators move around, Pauli webs are a good way to verify that the logical operators anti-commute with each other throughout all of the ISGs.

Using Pauli webs to explore codes where the logical operator is unknown is a powerful tool. While looking into variations of Floquet codes, using Pauli Webs on a ZX diagrams was a very effective method for finding logical operators and verifying they act correctly.

A.5 Concluding Points

ZX Calculus and overlaying Pauli webs offer powerful tools to analyze quantum error correcting circuits. It is especially helpful when analyzing codes that have non-trivial time dynamics like the Floquet Code. It offers a different visualization of commutation relations that offer a time sensitive point of view. In the work contributing to this thesis, they have been very helpful in finding missing detector coverage, finding logical operators for different measurement schedules, and realizing the relation between Pauli frames and anyon type in the Floquet section, prompting the switch to a CSS-style of measurement schedule.

B Floquet Matching Graphs

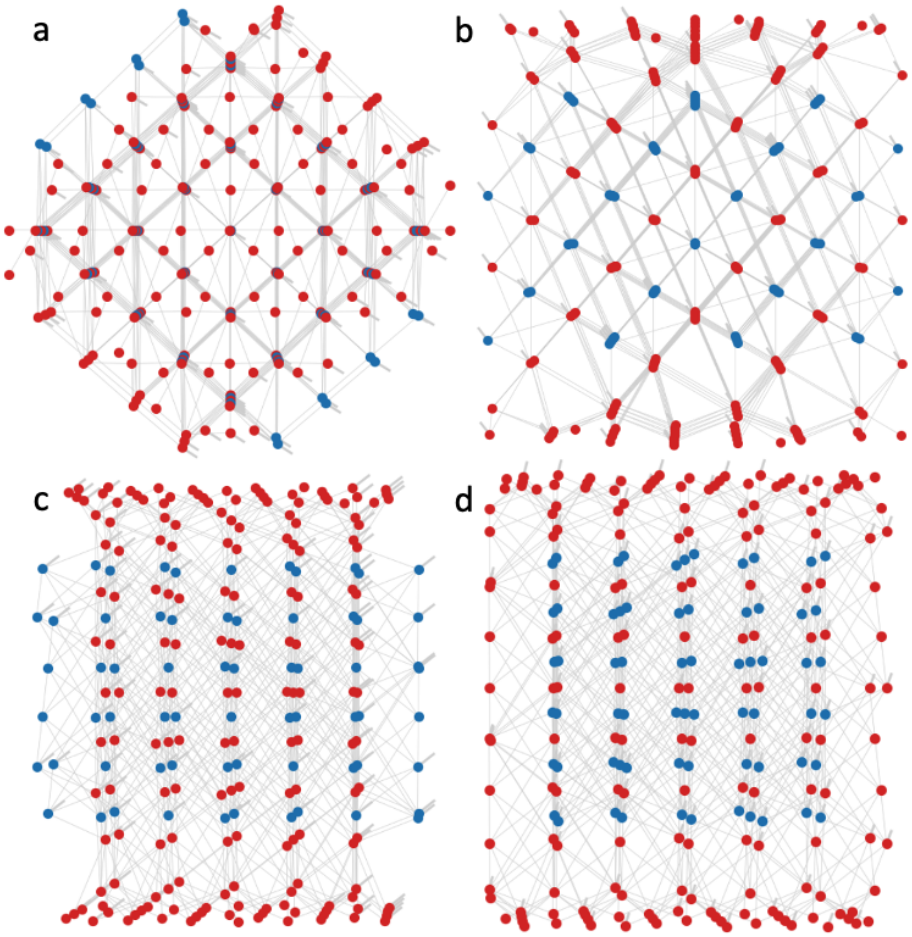


Figure 36: Here is the matching graph for a 5x5 distance-8 planar Floquet code from 4 different angles. Angle a) shows the matching graph from the top, viewing in the time dimension, while angles b), c), and d) show the matching graph from the three different spatial angles that line up with the hexagonal lattice. The top and bottom time-like detectors are much more dense than other layers because the initialization and readout detectors are created from individual checks themselves, leading to many small detectors at the time-like borders. In a) these individual detectors can be seen to line up with space-like edges associated with the links. Also notice how c) and d) line up such that the boundary plaquettes of the different anyon types can be seen sticking out on the sides.

C More Simulation and Real Device Data

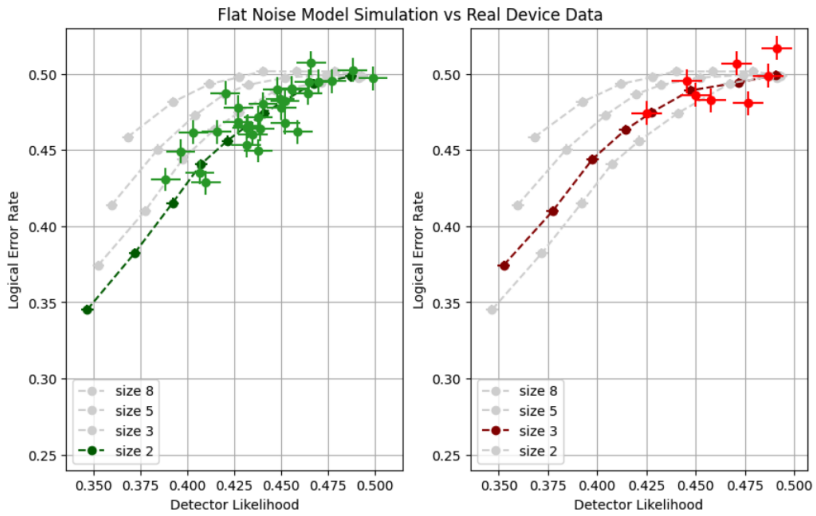


Figure 37: Data shown overlaying real device data and simulation data for 6 sub-rounds as a function of average detector likelihood. Considering how fast the data is rising to pure noise from the 4 sub-round image below, it is not worth showing 8 or more sub-round data sets because it would just be staring at noise.

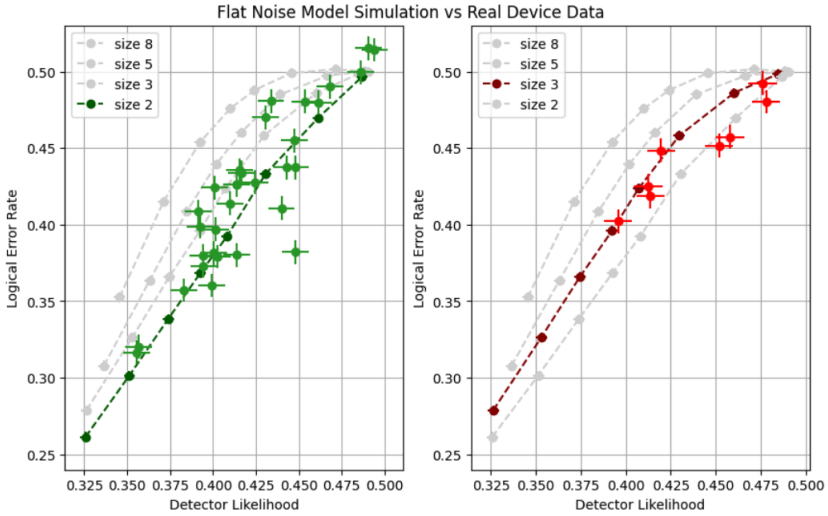


Figure 38: This is just a repeat of Figure 25 (4 sub-rounds) for comparison with the image above.

D Blank 2x2 Planar Floquet ZX-diagram

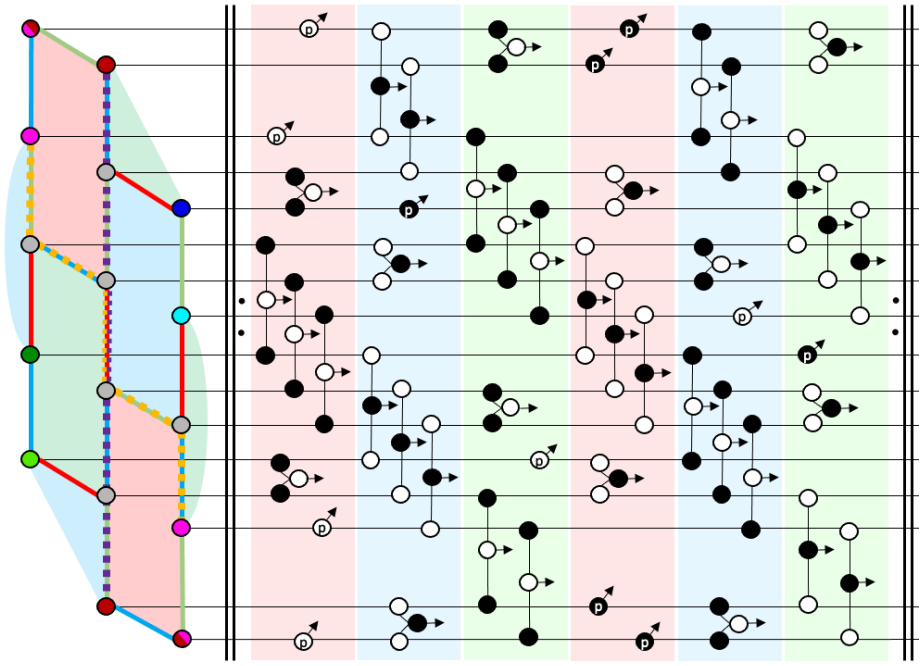


Figure 39: Here is a blank ZX diagram for a 2x2 distance-4 CSS-style planar Floquet code as defined in the main text. This is the same image as Figure 15, just without the Pauli webs overlaid. This is included in case the reader would like to try their hand at drawing their own Pauli webs. The logical operators are shown in the main text, but there are several detectors that were not shown and could be found here.

E Weather Report Circuit

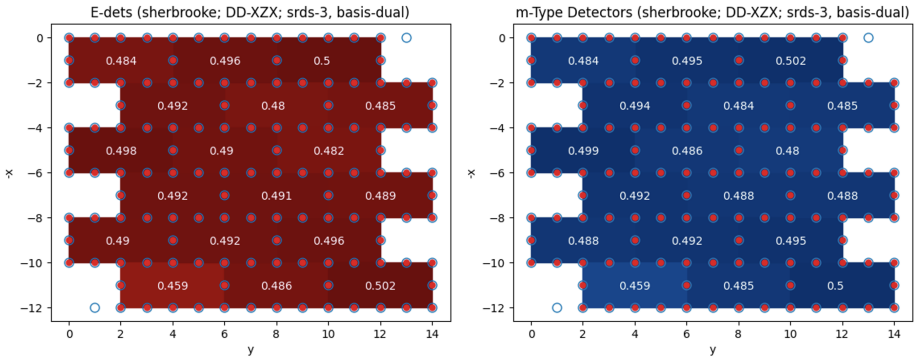


Figure 40: Above are the results for what we refer to as a weather report circuit. It is a Floquet code circuit run without any boundary truncated links or plaquettes run for 3 complete ISG cycles. The purpose of this circuit was originally intended to find bulk detector likelihoods, ideally finding the best locations and dynamical decoupling schemes to run actual Floquet codes. The issue is that these 125-qubit circuits ended up being so noisy that they were unusable. The above image shows the best dynamical decoupling scheme yielding the best results. The noise levels appear to be a hair under coin flips for almost everywhere. We include this in the appendix because it is evidence that the noise of a device increases with the size of the circuit being run, likely due to noise sources not captured in the standard circuit level Pauli noise model like increased cross-talk. Luckily, because the effective-p value is obtained by looking at the detector likelihood, we believe that it should include these other noise sources in the benchmarking scheme. The down side of this result is that the effective-p value will likely be variable in relation to code size run due to the changing of these other noise sources like increased cross-talk.

References

[Nis81] Hidetoshi Nishimori. "Internal Energy, Specific Heat and Correlation Function of the Bond-Random Ising Model". In: *Progress of Theoretical Physics* 66.4 (Oct. 1981). _eprint: https://academic.oup.com/ptp/article-pdf/66/4/1169/5265369/66-4-1169.pdf, pp. 1169–1181. ISSN: 0033-068X. DOI: 10.1143/PTP.66.1169. URL: https://doi.org/10.1143/PTP.66.1169.

[Got97] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. May 28, 1997. arXiv: quant-ph/9705052. URL: http://arxiv.org/abs/quant-ph/9705052 (visited on 01/06/2023).

[Den+02] Eric Dennis et al. "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. arXiv: quant-ph/0110143. URL: http://arxiv.org/abs/quant-ph/0110143 (visited on 01/06/2023).

[Kit03] A. Yu Kitaev. "Fault-tolerant quantum computation by anyons". In: *Annals of Physics* 303.1 (Jan. 2003), pp. 2–30. ISSN: 00034916. DOI: 10.1016/S0003-4916(02)00018-0. arXiv: quant-ph/9707021. URL: http://arxiv.org/abs/quant-ph/9707021 (visited on 01/06/2023).

[Pou05] David Poulin. "Stabilizer Formalism for Operator Quantum Error Correction". In: *Physical Review Letters* 95.23 (Dec. 1, 2005), p. 230504. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.95.230504. arXiv: quant-ph/0508131. URL: http://arxiv.org/abs/quant-ph/0508131 (visited on 04/18/2023).

[Kit06] Alexei Kitaev. "Anyons in an exactly solved model and beyond". In: *Annals of Physics* 321.1 (Jan. 2006), pp. 2–111. ISSN: 00034916. DOI: 10.1016/j.aop.2005.10.005. arXiv: cond-mat/0506438. URL: http://arxiv.org/abs/cond-mat/0506438 (visited on 07/03/2023).

[FWH12] Austin G. Fowler, Adam C. Whiteside, and Lloyd C. L. Hollenberg. "Towards practical classical processing for the surface code". In: *Physical Review Letters* 108.18 (May 1, 2012), p. 180501. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.108.180501. arXiv: 1110.5133[quant-ph]. URL: http://arxiv.org/abs/1110.5133 (visited on 01/06/2023).

[Fow14] Austin G. Fowler. *Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time*. Oct. 10, 2014. arXiv: 1307.1740[quant-ph]. URL: http://arxiv.org/abs/1307.1740 (visited on 01/06/2023).

[TS14] Yu Tomita and Krysta M. Svore. "Low-distance Surface Codes under Realistic Quantum Noise". In: *Physical Review A* 90.6 (Dec. 11, 2014), p. 062320. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.90.062320. arXiv: 1404.3747[quant-ph]. URL: http://arxiv.org/abs/1404.3747 (visited on 10/26/2023).

[BPW17] Miriam Backens, Simon Perdrix, and Quanlong Wang. "A Simplified Stabilizer ZX-calculus". In: *Electronic Proceedings in Theoretical Computer Science* 236 (Jan. 1, 2017), pp. 1–20. ISSN: 2075-2180. DOI: 10.4204/EPTCS.236.1. arXiv: 1602.04744[quant-ph]. URL: http://arxiv.org/abs/1602.04744 (visited on 04/25/2023).

[Bro+17] Benjamin J. Brown et al. "Poking holes and cutting corners to achieve Clifford gates with the surface code". In: *Physical Review X* 7.2 (May 24, 2017), p. 021029. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.7.021029. arXiv: 1609.04673[cond-mat,physics:quant-ph]. URL: http://arxiv.org/abs/1609.04673 (visited on 02/01/2023).

[FG19] Austin G. Fowler and Craig Gidney. *Low overhead quantum computation using lattice surgery*. Aug. 30, 2019. arXiv: 1808.06709[quant-ph]. URL: http://arxiv.org/abs/1808.06709 (visited on 01/06/2023).

[BH20] Niel de Beaudrap and Dominic Horsman. "The ZX calculus is a language for surface code lattice surgery". In: *Quantum* 4 (Jan. 9, 2020), p. 218. ISSN: 2521-327X. DOI: 10.22331/q-2020-01-09-218. arXiv: 1704.08670[quant-ph]. URL: http://arxiv.org/abs/1704.08670 (visited on 04/26/2023).

[Del20] Nicolas Delfosse. *Hierarchical decoding to reduce hardware requirements for quantum computing*. Jan. 30, 2020. arXiv: 2001.11427[quant-ph]. URL: http://arxiv.org/abs/2001.11427 (visited on 01/06/2023).

[Dun+20] Ross Duncan et al. "Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus". In: *Quantum* 4 (June 4, 2020), p. 279. ISSN: 2521-327X. DOI: 10.22331/q-2020-06-04-279. arXiv: 1902.03178[quant-ph]. URL: http://arxiv.org/abs/1902.03178 (visited on 05/01/2023).

[Wet20] John van de Wetering. *ZX-calculus for the working quantum computer scientist*. Dec. 27, 2020. arXiv: 2012.13966[quant-ph]. URL: http://arxiv.org/abs/2012.13966 (visited on 06/08/2023).

[Bac+21] Miriam Backens et al. “There and back again: A circuit extraction tale”. In: *Quantum* 5 (Mar. 25, 2021), p. 421. ISSN: 2521-327X. DOI: 10.22331/q-2021-03-25-421. arXiv: 2003.01664[quant-ph]. URL: http://arxiv.org/abs/2003.01664 (visited on 05/01/2023).

[Che+21] Zijun Chen et al. “Exponential suppression of bit or phase flip errors with repetitive error correction”. In: *Nature* 595.7867 (July 15, 2021), pp. 383–387. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-021-03588-y. arXiv: 2102.06132[quant-ph]. URL: http://arxiv.org/abs/2102.06132 (visited on 01/06/2023).

[DN21] Nicolas Delfosse and Naomi H. Nickerson. “Almost-linear time decoding algorithm for topological codes”. In: *Quantum* 5 (Dec. 2, 2021), p. 595. ISSN: 2521-327X. DOI: 10.22331/q-2021-12-02-595. arXiv: 1709.06218[quant-ph]. URL: http://arxiv.org/abs/1709.06218 (visited on 01/06/2023).

[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. arXiv: 2103.02202[quant-ph]. URL: http://arxiv.org/abs/2103.02202 (visited on 01/06/2023).

[Gid+21] Craig Gidney et al. “A Fault-Tolerant Honeycomb Memory”. In: *Quantum* 5 (Dec. 20, 2021), p. 605. ISSN: 2521-327X. DOI: 10.22331/q-2021-12-20-605. arXiv: 2108.10457[quant-ph]. URL: http://arxiv.org/abs/2108.10457 (visited on 01/06/2023).

[HH21] Matthew B. Hastings and Jeongwan Haah. “Dynamically Generated Logical Qubits”. In: *Quantum* 5 (Oct. 19, 2021), p. 564. ISSN: 2521-327X. DOI: 10.22331/q-2021-10-19-564. arXiv:

2107.02194[quant-ph]. URL: http://arxiv.org/abs/2107.02194 (visited on 01/06/2023).

[Hig21] Oscar Higgott. *PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching*. July 12, 2021. arXiv: 2105.13082[quant-ph]. URL: http://arxiv.org/abs/2105.13082 (visited on 01/06/2023).

[Ach+22] Rajeev Acharya et al. *Suppressing quantum errors by scaling a surface code logical qubit*. July 20, 2022. arXiv: 2207.06431[quant-ph]. URL: http://arxiv.org/abs/2207.06431 (visited on 01/06/2023).

[And+22] Trond I. Andersen et al. *Observation of non-Abelian exchange statistics on a superconducting processor*. Oct. 18, 2022. arXiv: 2210.10255[cond-mat,physics:quant-ph]. URL: http://arxiv.org/abs/2210.10255 (visited on 05/15/2023).

[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. arXiv: 2202.11845[quant-ph]. URL: http://arxiv.org/abs/2202.11845 (visited on 01/06/2023).

[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. arXiv: 2110.09545[quant-ph]. URL: http://arxiv.org/abs/2110.09545 (visited on 04/18/2023).

[Kes+22] Markus S. Kesselring et al. *Anyon condensation and the color code*. Nov. 30, 2022. arXiv: 2212.00042[cond-mat,physics:

quant-ph]. URL: http://arxiv.org/abs/2212.00042 (visited on 04/26/2023).

[Kis22] Aleks Kissinger. *Phase-free ZX diagrams are CSS codes (...or how to graphically grok the surface code)*. Apr. 29, 2022. arXiv: 2204.14038[quant-ph]. URL: http://arxiv.org/abs/2204.14038 (visited on 04/25/2023).

[Kri+22] Sebastian Krinner et al. "Realizing Repeated Quantum Error Correction in a Distance-Three Surface Code". In: *Nature* 605.7911 (May 26, 2022), pp. 669–674. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-022-04566-8. arXiv: 2112.03708[cond-mat, physics:quant-ph]. URL: http://arxiv.org/abs/2112.03708 (visited on 01/06/2023).

[Lee+22] Jong Yeon Lee et al. *Decoding Measurement-Prepared Quantum Phases and Transitions: from Ising model to gauge theory, and beyond*. Sept. 6, 2022. arXiv: 2208.11699[cond-mat,physics:quant-ph]. URL: http://arxiv.org/abs/2208.11699 (visited on 10/23/2023).

[Woo22a] James R. Wootton. "Hexagonal matching codes with 2-body measurements". In: *Journal of Physics A: Mathematical and Theoretical* 55.29 (July 22, 2022), p. 295302. ISSN: 1751-8113, 1751-8121. DOI: 10.1088/1751-8121/ac7a75. arXiv: 2109.13308[quant-ph]. URL: http://arxiv.org/abs/2109.13308 (visited on 04/18/2023).

[Woo22b] James R. Wootton. *Measurements of Floquet code plaquette stabilizers*. Oct. 24, 2022. arXiv: 2210.13154[quant-ph]. URL: http://arxiv.org/abs/2210.13154 (visited on 04/27/2023).

[WLZ22] Yue Wu, Namitha Liyanage, and Lin Zhong. *An interpretation of Union-Find Decoder on Weighted Graphs*. Nov. 6, 2022. arXiv: 2211.03288[quant-ph]. URL: http://arxiv.org/abs/2211.03288 (visited on 01/26/2023).

[Aas+23] David Aasen et al. *Fault-Tolerant Hastings-Haah Codes in the Presence of Dead Qubits*. July 7, 2023. arXiv: 2307.03715[quant-ph]. URL: http://arxiv.org/abs/2307.03715 (visited on 07/10/2023).

[Bom+23a] Hector Bombin et al. "Logical blocks for fault-tolerant topological quantum computation". In: *PRX Quantum* 4.2 (Apr. 7, 2023), p. 020303. ISSN: 2691-3399. DOI: 10.1103/PRXQuantum.4.020303. arXiv: 2112.12160[quant-ph]. URL: http://arxiv.org/abs/2112.12160 (visited on 04/26/2023).

[Bom+23b] Hector Bombin et al. *Unifying flavors of fault tolerance with the ZX calculus*. Mar. 15, 2023. arXiv: 2303.08829[quant-ph]. URL: http://arxiv.org/abs/2303.08829 (visited on 04/18/2023).

[Che+23] Edward H. Chen et al. *Realizing the Nishimori transition across the error threshold for constant-depth quantum circuits*. Sept. 6, 2023. arXiv: 2309.02863[cond-mat,physics:quant-ph]. URL: http://arxiv.org/abs/2309.02863 (visited on 09/14/2023).

[Coe23] Bob Coecke. *Basic ZX-calculus for students and professionals*. Mar. 6, 2023. arXiv: 2303.03163[quant-ph]. URL: http://arxiv.org/abs/2303.03163 (visited on 04/25/2023).

[ESD23] Tyler D. Ellison, Joseph Sullivan, and Arpit Dua. *Floquet codes with a twist*. June 13, 2023. arXiv: 2306.08027[quant-ph]. URL: http://arxiv.org/abs/2306.08027 (visited on 06/16/2023).

[GHU23] Spiro Gicev, Lloyd C. L. Hollenberg, and Muhammad Usman. *Quantum computer error structure probed by quantum error correction syndrome measurements*. Oct. 18, 2023. arXiv: 2310.12448[quant-ph]. URL: http://arxiv.org/abs/2310.12448 (visited on 10/20/2023).

[Gid23] Craig Gidney. *Inplace Access to the Surface Code Y Basis*. Feb. 14, 2023. arXiv: 2302.07395[quant-ph]. URL: http://arxiv.org/abs/2302.07395 (visited on 05/24/2023).

[HW23] Bence Hetényi and James R. Wootton. *Tailoring quantum error correction to spin qubits*. June 30, 2023. arXiv: 2306.17786[quant-ph]. URL: http://arxiv.org/abs/2306.17786 (visited on 08/21/2023).

[Liy+23] Namitha Liyanage et al. *Scalable Quantum Error Correction for Surface Codes using FPGA*. Jan. 19, 2023. arXiv: 2301.08419[quant-ph]. URL: http://arxiv.org/abs/2301.08419 (visited on 01/26/2023).

[MBG23] Matt McEwen, Dave Bacon, and Craig Gidney. *Relaxing Hardware Requirements for Surface Code Circuits using Time-dynamics*. Feb. 4, 2023. arXiv: 2302.02192[quant-ph]. URL: http://arxiv.org/abs/2302.02192 (visited on 04/18/2023).

[Pae+23] Adam Paetznick et al. "Performance of planar Floquet codes with Majorana-based qubits". In: *PRX Quantum* 4.1 (Jan. 25, 2023), p. 010310. ISSN: 2691-3399. DOI: 10.1103/PRXQuantum.4.010310. arXiv: 2202.11829[quant-ph]. URL: http://arxiv.org/abs/2202.11829 (visited on 06/28/2023).

[SBB23] Armands Strikis, Simon C. Benjamin, and Benjamin J. Brown. *Quantum computing is scalable on a planar array of qubits with*

fabrication defects. June 28, 2023. arXiv: 2111.06432[cond-mat, physics:quant-ph]. URL: http://arxiv.org/abs/2111.06432 (visited on 07/10/2023).

[Wan23] Quanlong Wang. *Completeness of the ZX-calculus*. May 17, 2023. arXiv: 2209.14894[quant-ph]. URL: http://arxiv.org/abs/2209.14894 (visited on 09/27/2023).