

# QEC1

## A Mathematical Exposition

Auto-generated from Lean 4 Formalization

April 13, 2026

### Abstract

This document presents the mathematical content from the `QEC1` library. The material has been translated from a verified Lean 4 formalization, ensuring mathematical rigor while maintaining readability.

#### **WARNING: Unproven Axioms**

This formalization uses the following unproven axiom(s): `ZeAvMeas_isGaugingStabilizer`, `generators_span_all_detectors`, `initXe_isGaugingStabilizer`, `readoutXe_isGaugingStabilizer`, `space_fault_cleaning`, `spacetimeStabilizer_completeness`, `syndromeFree_pureSpace_inCentralizer`, `timePropagating_isGaugingStabilizer`.

These axioms were introduced because the full proofs were not completed. The mathematical validity of results depends on these assumptions.

## Contents

<b>1</b>	<b>Mathematical Content</b>	<b>3</b>
1.1	Remark 1: NotationBinaryVectors . . . . .	3
1.2	Remark 2: NotationPauliOperators . . . . .	4
1.3	Remark 3: NotationStabilizerCodes . . . . .	5
1.4	Remark 4: NotationCheegerConstant . . . . .	6
1.5	Definition 1: BoundaryAndCoboundaryMaps . . . . .	8
1.6	Remark 5: ExactnessOfSequences . . . . .	10
1.7	Definition 2: GaussLawAndFluxOperators . . . . .	13
1.8	Definition 3: DeformedOperator . . . . .	14
1.9	Remark 6: NoncommutingOperatorsCannotBeDeformed . . . . .	17
1.10	Definition 4: DeformedCode . . . . .	20
1.11	Statement : QEC1 . . . . .	21
1.12	Lemma 1: DeformedCodeChecks . . . . .	22
1.13	Remark 7: CodespaceDimensionAfterGauging . . . . .	23
1.14	Remark 8: FreedomInDeformedChecks . . . . .	24
1.15	Definition 5: GaugingMeasurementAlgorithm . . . . .	26
1.16	Theorem 1: GaugingMeasurementCorrectness . . . . .	27
1.17	Remark 9: CircuitImplementation . . . . .	28
1.18	Remark 10: FlexibilityOfGraphG . . . . .	29
1.19	Remark 11: DesiderataForGraphG . . . . .	32
1.20	Definition 6: CycleSparsifiedGraph . . . . .	33
1.21	Lemma 2: DecongestionLemmaBound . . . . .	34

1.22 Remark 12: WorstCaseGraphConstruction . . . . .	35
1.23 Lemma 3: SpaceDistance . . . . .	37
1.24 Remark 13: OptimalCheegerConstant . . . . .	39
1.25 Remark 14: ParallelGaugingMeasurement . . . . .	42
1.26 Definition 7: SpaceAndTimeFaults . . . . .	42
1.27 Definition 8: Detectors . . . . .	43
1.28 Definition 9: Syndrome . . . . .	47
1.29 Definition 10: FaultTolerantGaugingProcedure . . . . .	49
1.30 Lemma 4: SpacetimeCodeDetectors . . . . .	50
1.31 Definition 11: SpacetimeLogicalFault . . . . .	52
1.32 Definition 12: SpacetimeFaultDistance . . . . .	53
1.33 Lemma 5: SpacetimeStabilizers . . . . .	55
1.34 Lemma 6: TimeFaultDistance . . . . .	56
1.35 Lemma 7: SpaceTimeDecoupling . . . . .	58
1.36 Theorem 2: FaultTolerantGaugingDistance . . . . .	60
1.37 Remark 15: FluxCheckMeasurementFrequency . . . . .	61
1.38 Remark 16: BoundaryRoundsOverkill . . . . .	63
1.39 Corollary 1: WorstCaseOverhead . . . . .	64
1.40 Remark 17: HypergraphGeneralization . . . . .	65
1.41 Remark 18: RelationToLatticeSurgery . . . . .	66
1.42 Remark 19: BivariateBicycleCodeNotation . . . . .	69
1.43 Remark 20: GrossCodeDefinition . . . . .	70
1.44 Remark 21: GrossCodeGaugingMeasurement . . . . .	71
1.45 Remark 22: DoubleGrossCodeDefinition . . . . .	71
1.46 Remark 23: GeneralizationsBeyondPauli . . . . .	72
1.47 Remark 24: ShorStyleMeasurement . . . . .	74
1.48 Remark 25: SteaneStyleMeasurement . . . . .	75
1.49 Remark 26: CohenEtAlSchemeRecovery . . . . .	75

# 1 Mathematical Content

## 1.1 Remark 1: Notation Binary Vectors

Binary vectors over the field  $\mathbb{F}_2 = \mathbb{Z}_2$  provide a natural algebraic framework for representing combinatorial structures in graph theory. By encoding subsets of vertices, edges, or cycles as characteristic functions taking values in  $\mathbb{F}_2$ , we can leverage the algebraic properties of this field to perform set-theoretic operations. This identification proves particularly powerful because addition in  $\mathbb{F}_2$  corresponds precisely to the symmetric difference of sets, transforming combinatorial problems into linear algebra over finite fields.

The key insight is that every subset  $S$  of a finite set  $\alpha$  can be uniquely represented by its characteristic vector  $\chi_S : \alpha \rightarrow \mathbb{F}_2$ , where  $\chi_S(v) = 1$  if and only if  $v \in S$ . This correspondence preserves the essential structure: symmetric difference of sets becomes vector addition, intersection relates to pointwise multiplication, and complement corresponds to adding the all-ones vector. This algebraic perspective enables the use of linear algebraic techniques in combinatorial settings.

*Remark* (Remark 1: Notation for Binary Vectors). Throughout this work we use binary vectors over  $\mathbb{F}_2$  to indicate collections of vertices, edges, and cycles of a graph  $G$ . We identify the binary vector associated to a set of vertices, edges, or cycles with the set itself. Addition of binary vectors corresponds to symmetric difference of sets.

This identification between sets and their characteristic vectors is formalized through the following definition, which establishes the fundamental correspondence between combinatorial and algebraic structures.

**Definition** (Definition: Characteristic Vector). Let  $\alpha$  be a type. The **characteristic vector** of a set  $S \subseteq \alpha$  is the function  $\chi_S : \alpha \rightarrow \mathbb{F}_2$  defined by

$$\chi_S(v) = \begin{cases} 1 & \text{if } v \in S, \\ 0 & \text{otherwise.} \end{cases}$$

The basic properties of characteristic vectors establish the precise correspondence between set membership and function values. These fundamental results show that the characteristic vector encoding is both faithful and computationally tractable.

**Theorem** (Membership Characterization). *Let  $S \subseteq \alpha$  and  $v \in \alpha$ . Then  $\chi_S(v) = 1$  if and only if  $v \in S$ .*

*Proof.* We prove both directions separately.

( $\Rightarrow$ ): Assume  $\chi_S(v) = 1$ . Suppose for contradiction that  $v \notin S$ . Then by definition of the characteristic vector,  $\chi_S(v) = 0$ , which contradicts our hypothesis since  $1 \neq 0$  in  $\mathbb{F}_2$ .

( $\Leftarrow$ ): If  $v \in S$ , then  $\chi_S(v) = 1$  follows directly from the definition of the characteristic vector.  $\square$

The most significant property is that symmetric difference of sets corresponds exactly to pointwise addition of their characteristic vectors in  $\mathbb{F}_2$ . This is the foundation for all algebraic manipulations of set operations.

**Theorem** (Symmetric Difference Corresponds to Addition). *For any sets  $S, T \subseteq \alpha$ ,*

$$\chi_{S \Delta T} = \chi_S + \chi_T,$$

*where addition is performed pointwise in  $\mathbb{F}_2$ .*

*Proof.* By extensionality, it suffices to show  $\chi_{S\Delta T}(v) = \chi_S(v) + \chi_T(v)$  for arbitrary  $v \in \alpha$ . We consider all possible cases based on membership in  $S$  and  $T$ .

**Case 1:**  $v \in S$  and  $v \in T$ . Then  $v \notin S\Delta T$  since symmetric difference excludes elements in both sets. Thus  $\chi_{S\Delta T}(v) = 0$  while  $\chi_S(v) = 1$  and  $\chi_T(v) = 1$ . The equality  $0 = 1 + 1$  holds in  $\mathbb{F}_2$ .

**Case 2:**  $v \in S$  and  $v \notin T$ . Then  $v \in S\Delta T$ , so  $\chi_{S\Delta T}(v) = 1$ . We have  $\chi_S(v) = 1$  and  $\chi_T(v) = 0$ , giving  $1 = 1 + 0$  in  $\mathbb{F}_2$ .

**Case 3:**  $v \notin S$  and  $v \in T$ . Then  $v \in S\Delta T$ , so  $\chi_{S\Delta T}(v) = 1$ . We have  $\chi_S(v) = 0$  and  $\chi_T(v) = 1$ , giving  $1 = 0 + 1$  in  $\mathbb{F}_2$ .

**Case 4:**  $v \notin S$  and  $v \notin T$ . Then  $v \notin S\Delta T$ , so all characteristic vectors evaluate to 0, giving  $0 = 0 + 0$ .  $\square$

This correspondence extends to other set operations. For instance, the union of sets can be expressed in terms of characteristic vectors, though the formula is more complex due to the overlap correction required in characteristic 2.

**Theorem** (Union in Terms of Characteristic Vectors). *For any sets  $S, T \subseteq \alpha$ ,*

$$\chi_{S\cup T} = \chi_S + \chi_T + \chi_{S\cap T},$$

where addition is performed pointwise in  $\mathbb{F}_2$ .

*Proof.* The proof follows the same case analysis as the symmetric difference theorem. The key observation is that when  $v \in S \cap T$ , we need to add the intersection term to correct for the fact that  $\chi_S(v) + \chi_T(v) = 1 + 1 = 0$  in  $\mathbb{F}_2$ , but we want  $\chi_{S\cup T}(v) = 1$ .  $\square$

The characteristic vector correspondence is injective, ensuring that distinct sets have distinct characteristic vectors. This injectivity is crucial for the identification between sets and binary vectors.

**Theorem** (Injectivity of Characteristic Vectors). *The map  $\chi : \mathcal{P}(\alpha) \rightarrow (\alpha \rightarrow \mathbb{F}_2)$  sending  $S \mapsto \chi_S$  is injective.*

*Proof.* Suppose  $\chi_S = \chi_T$  for sets  $S, T \subseteq \alpha$ . Then for any  $v \in \alpha$ , we have  $\chi_S(v) = \chi_T(v)$ . By the membership characterization,  $v \in S \Leftrightarrow \chi_S(v) = 1 \Leftrightarrow \chi_T(v) = 1 \Leftrightarrow v \in T$ . Thus  $S = T$  by set extensionality.  $\square$

This algebraic framework proves particularly valuable in graph theory, where we frequently work with sets of vertices, edges, and cycles. The ability to perform set operations through simple arithmetic in  $\mathbb{F}_2$  enables powerful computational and theoretical techniques that would be cumbersome in purely combinatorial approaches.

## 1.2 Remark 2: NotationPauliOperators

The study of quantum error correction requires a precise mathematical framework for describing and manipulating quantum operations on multi-qubit systems. Pauli operators form the foundation of this framework, as they generate the full operator algebra acting on qubits and provide a natural basis for understanding quantum errors and their correction.

In quantum computing, each qubit can be acted upon by one of four single-qubit Pauli operators: the identity  $I$ , bit-flip  $X$ , phase-flip  $Z$ , or combined  $Y = XZ$  (up to phase). For systems of multiple qubits, we consider tensor products of these operators. The key insight is that we can represent such operators algebraically using binary vectors, which greatly simplifies both theoretical analysis and computational implementation.

*Remark* (Remark 2: Notation for Pauli Operators). For qubits labeled by vertices  $v$  of a graph or indices  $i$ , we denote by  $X_v$  (or  $X_i$ ) the Pauli- $X$  operator acting on qubit  $v$  (or  $i$ ), and similarly  $Z_v$  (or  $Z_i$ ) for Pauli- $Z$ . A product of Pauli operators is written as  $\prod_{v \in S} X_v$  for a set  $S$  of qubit labels. The identity operator is denoted  $\mathbb{I}$ .

For a Pauli operator  $P$ , we denote by  $S_X(P)$  the  $X$ -type support (sites where  $P$  acts via  $X$  or  $Y$ ) and  $S_Z(P)$  the  $Z$ -type support (sites where  $P$  acts via  $Y$  or  $Z$ ). This binary representation captures the essential structure of Pauli operators while abstracting away phase factors, which is sufficient for most quantum error correction applications.

A Pauli operator on qubits labeled by a type  $V$  is represented as a pair of binary vectors  $(\mathbf{xVec}, \mathbf{zVec}) \in (\mathbb{Z}/2\mathbb{Z})^V \times (\mathbb{Z}/2\mathbb{Z})^V$ , where the pair  $(x, z)$  represents the operator  $\bigotimes_v X_v^{x_v} Z_v^{z_v}$ . At each site  $v$ :  $(0, 0)$  means the identity  $I$ ,  $(1, 0)$  means  $X$ ,  $(0, 1)$  means  $Z$ , and  $(1, 1)$  means  $Y$  (up to phase).

This notation establishes a clean algebraic framework that will prove essential for analyzing quantum error correction codes. The binary vector representation allows us to leverage the rich theory of linear algebra over finite fields, while the support notation provides geometric intuition about which qubits are affected by a given operation. The abstraction of phase factors is justified because most error correction protocols depend only on the Pauli structure, not on the precise phase relationships, making this representation both mathematically elegant and practically useful.

### 1.3 Remark 3: NotationStabilizerCodes

In quantum error correction, stabilizer codes represent a fundamental class of quantum codes where the code space is defined as the joint eigenspace of a set of commuting Pauli operators called stabilizers. Understanding the algebraic structure of these operators requires precise notation for commutation relations and the relationships between different types of operators in the code. The symplectic inner product provides the key tool for characterizing when Pauli operators commute, which is essential for defining valid stabilizer codes.

*Remark* (Remark 3: Notation for Stabilizer Codes). We establish the fundamental notation and terminology for stabilizer codes, building on the symplectic structure of Pauli operators. The symplectic inner product determines commutation relations in the Pauli group, while the stabilizer and centralizer structures define the code space and logical operators respectively.

The symplectic inner product between two Pauli operators  $P, Q : \text{PauliOp}(V)$  is defined as

$$\langle P, Q \rangle_{\text{symp}} = \sum_{v \in V} (P.x_v \cdot Q.z_v + P.z_v \cdot Q.x_v) \in \mathbb{Z}/2\mathbb{Z}.$$

Two Pauli operators  $P$  and  $Q$  commute in the full Pauli group if and only if their symplectic inner product vanishes:

$$\text{PauliCommute}(P, Q) \iff \langle P, Q \rangle_{\text{symp}} = 0.$$

A stabilizer code on qubits labeled by finite type  $V$  consists of a finite index set  $I$  of stabilizer checks, a map  $\text{check} : I \rightarrow \text{PauliOp}(V)$ , and the requirement that all checks mutually commute:  $\text{PauliCommute}(\text{check}(i), \text{check}(j))$  for all  $i, j \in I$ .

The stabilizer group  $S$  is the subgroup generated by the checks:

$$S = \langle \text{check}(i) \mid i \in I \rangle \leq \text{PauliOp}(V).$$

The centralizer consists of all Pauli operators that commute with every stabilizer check:

$$C(S) = \{P \in \text{PauliOp}(V) \mid \forall i \in I, \text{PauliCommute}(P, \text{check}(i))\}.$$

A non-trivial logical operator is a Pauli operator in the centralizer that is neither the identity nor an element of the stabilizer group.

This notation provides the foundation for analyzing quantum error correction codes. The symplectic inner product captures the essential algebraic structure, while the distinction between stabilizer elements and logical operators determines the code’s error-correcting capabilities. The parameters  $\llbracket n, k, d \rrbracket$  encode the number of physical qubits  $n$ , logical qubits  $k = n - |\text{numChecks}(C)|$ , and minimum distance  $d$  of the code.

#### 1.4 Remark 4: NotationCheegerConstant

The Cheeger constant provides a fundamental measure of a graph’s connectivity and expansion properties. It quantifies how well-connected a graph is by measuring the minimum ratio of edge boundary to subset size across all balanced cuts. This concept is central to the theory of expander graphs, which have applications ranging from computer science to coding theory.

*Remark* (Remark 4: Cheeger Constant and Expander Graphs). For a finite simple graph  $G = (V, E)$ , the **Cheeger constant** (also called isoperimetric number or edge expansion) is defined as

$$h(G) = \min_{\substack{S \subseteq V \\ 0 < |S| \leq |V|/2}} \frac{|\partial S|}{|S|},$$

where  $\partial S = \{e = \{u, v\} \in E : u \in S, v \notin S\}$  is the edge boundary of  $S$ . A graph with  $h(G) \geq c$  for some constant  $c > 0$  is called a  **$c$ -expander graph**.

The edge boundary captures the notion of a "cut" in graph theory, measuring how many edges must be removed to separate a subset from its complement. The restriction to subsets of size at most  $|V|/2$  ensures we consider balanced cuts, which are typically the bottlenecks for connectivity. Expander graphs maintain high connectivity even for small cuts, making them valuable in applications requiring robust communication networks.

**Definition** (Edge Boundary). The *edge boundary* of a vertex subset  $S$  in a simple graph  $G = (V, E)$  is the set of edges with exactly one endpoint in  $S$ . Formally,

$$\partial S := \{e \in E : |e_{\text{set}} \cap S| = 1\},$$

where  $e_{\text{set}}$  denotes the underlying two-element set of the edge  $e$ .

**Theorem** (Membership in Edge Boundary). *An edge  $e$  belongs to the edge boundary  $\partial S$  if and only if  $e \in E$  and  $|e_{\text{set}} \cap S| = 1$ .*

*Proof.* This holds by definition, since  $\partial S$  is defined as the filter of the edge set by the condition  $|e_{\text{set}} \cap S| = 1$ . The equivalence follows directly from unfolding the definition of edge boundary.  $\square$

**Theorem** (Edge Boundary of Empty Set). *The edge boundary of the empty set is empty:  $\partial \emptyset = \emptyset$ .*

*Proof.* By extensionality, it suffices to show that no edge  $e$  belongs to  $\partial \emptyset$ . For any edge  $e$ , we have  $e_{\text{set}} \cap \emptyset = \emptyset$ , so  $|e_{\text{set}} \cap \emptyset| = 0 \neq 1$ . Therefore, the cardinality condition for membership in the edge boundary is never satisfied.  $\square$

**Theorem** (Edge Boundary of Universal Set). *The edge boundary of the entire vertex set is empty:  $\partial V = \emptyset$ .*

*Proof.* Suppose  $e \in \partial V$ . Then  $e \in E$  and  $|e_{\text{set}} \cap V| = 1$ . Since  $e$  is an edge in a simple graph, it connects two distinct vertices, so  $|e_{\text{set}}| = 2$ . However,  $e_{\text{set}} \cap V = e_{\text{set}}$  since all vertices of  $e$  lie in  $V$ , giving  $|e_{\text{set}}| = 1$ . This contradicts  $|e_{\text{set}}| = 2$ . Therefore,  $\partial V = \emptyset$ .  $\square$

**Theorem** (Edge Boundary of Complement). *The edge boundary of a set equals the edge boundary of its complement:  $\partial S = \partial S^c$ .*

*Proof.* We show both inclusions using the membership characterization. Let  $e \in E$  with  $|e_{\text{set}}| = 2$  (since  $e$  is a non-diagonal edge).

For the forward direction, suppose  $|e_{\text{set}} \cap S| = 1$ . Since  $e_{\text{set}} = (e_{\text{set}} \cap S) \cup (e_{\text{set}} \cap S^c)$  and these sets are disjoint, we have

$$|e_{\text{set}}| = |e_{\text{set}} \cap S| + |e_{\text{set}} \cap S^c|.$$

Substituting  $|e_{\text{set}}| = 2$  and  $|e_{\text{set}} \cap S| = 1$  gives  $|e_{\text{set}} \cap S^c| = 1$ .

For the reverse direction, suppose  $|e_{\text{set}} \cap S^c| = 1$ . Using the same partition identity and noting that  $(S^c)^c = S$ , we obtain  $|e_{\text{set}} \cap S| = 2 - 1 = 1$ .  $\square$

**Definition** (Cheeger-Valid Subsets). The set of *Cheeger-valid subsets* of a finite vertex set  $V$  is the collection of all nonempty subsets  $S \subseteq V$  satisfying  $2|S| \leq |V|$ :

$$\mathcal{S}(V) := \{S \subseteq V : S \neq \emptyset \text{ and } 2|S| \leq |V|\}.$$

**Definition** (Cheeger Constant). The *Cheeger constant*  $h(G)$  of a finite simple graph  $G$  is defined as

$$h(G) := \begin{cases} \inf_{S \in \mathcal{S}(V)} \frac{|\partial S|}{|S|} & \text{if } \mathcal{S}(V) \neq \emptyset, \\ 0 & \text{otherwise (i.e., when } |V| \leq 1). \end{cases}$$

**Definition** (Expander Graph). A graph  $G$  is a *c-expander* if  $0 < c$  and  $c \leq h(G)$ .

**Theorem** (Existence of Cheeger-Valid Subsets). *If  $|V| \geq 2$ , then the set of Cheeger-valid subsets  $\mathcal{S}(V)$  is nonempty.*

*Proof.* Since  $|V| \geq 2 > 0$ , there exists some vertex  $v \in V$ . The singleton  $\{v\}$  satisfies the required conditions: it is nonempty and  $2 \cdot 1 = 2 \leq |V|$ . Therefore,  $\{v\} \in \mathcal{S}(V)$ .  $\square$

**Theorem** (Cheeger Constant is Non-negative). *For any finite simple graph  $G$ , we have  $h(G) \geq 0$ .*

*Proof.* If  $\mathcal{S}(V)$  is nonempty, then  $h(G)$  is the infimum of ratios  $|\partial S|/|S|$  where  $S \in \mathcal{S}(V)$ . Each ratio is non-negative since both  $|\partial S| \geq 0$  and  $|S| > 0$  (as  $S$  is nonempty). Therefore, the infimum is also non-negative. If  $\mathcal{S}(V)$  is empty, then  $h(G) = 0 \geq 0$  by definition.  $\square$

**Theorem** (Edge Boundary Lower Bound from Cheeger Constant). *Let  $G$  be a finite simple graph. If  $c \leq h(G)$ ,  $S$  is a nonempty subset of  $V$ , and  $2|S| \leq |V|$ , then  $c \cdot |S| \leq |\partial S|$ .*

*Proof.* The conditions on  $S$  ensure that  $S \in \mathcal{S}(V)$ , so  $\mathcal{S}(V)$  is nonempty. Since  $S$  is nonempty,  $|S| > 0$ . By definition of the infimum,  $h(G) \leq |\partial S|/|S|$ . Therefore:

$$c \cdot |S| \leq h(G) \cdot |S| \leq \frac{|\partial S|}{|S|} \cdot |S| = |\partial S|.$$

$\square$

**Theorem** (Characterization of Expander Graphs). *Let  $G$  be a finite simple graph with  $|V| \geq 2$ . Then  $G$  is a  $c$ -expander if and only if  $0 < c$  and for every nonempty subset  $S \subseteq V$  with  $2|S| \leq |V|$ , we have  $c \cdot |S| \leq |\partial S|$ .*

*Proof. Forward direction.* If  $G$  is a  $c$ -expander, then  $0 < c$  and  $c \leq h(G)$ . For any nonempty  $S$  with  $2|S| \leq |V|$ , the previous theorem gives  $c \cdot |S| \leq |\partial S|$ .

**Reverse direction.** Suppose  $0 < c$  and the edge boundary condition holds for all valid  $S$ . Since  $|V| \geq 2$ , we have  $\mathcal{S}(V) \neq \emptyset$ . To show  $c \leq h(G)$ , we verify that  $c$  is a lower bound for all ratios  $|\partial S|/|S|$  with  $S \in \mathcal{S}(V)$ . For such  $S$ , we have  $|S| > 0$ , so the hypothesis gives  $c \leq |\partial S|/|S|$ . Taking the infimum over all valid  $S$  yields  $c \leq h(G)$ .  $\square$

This characterization provides a practical way to verify the expander property: instead of computing the Cheeger constant directly, one can check the edge boundary condition for all balanced cuts.

## 1.5 Definition 1: BoundaryAndCoboundaryMaps

In algebraic topology and homological algebra, the study of cell complexes requires systematic ways to relate cells of different dimensions. When working with graphs viewed as 1-dimensional cell complexes, we need linear maps that capture the incidence relationships between vertices and edges. These maps, known as boundary and coboundary operators, form the foundation of the chain complex structure that allows us to compute homology groups and understand the topological properties of the graph.

The boundary map measures how edges are incident to vertices, while the coboundary map works in the opposite direction, encoding how vertex functions extend to edge functions. Working over  $\mathbb{F}_2 = \mathbb{Z}_2$ , these maps have particularly clean algebraic properties that make them especially useful in applications such as quantum error correction and coding theory.

**Definition** (Definition 1: Boundary Map). Let  $G = (V, E)$  be a finite simple graph. The **boundary map**  $\partial : \mathbb{F}_2^E \rightarrow \mathbb{F}_2^V$  is the  $\mathbb{F}_2$ -linear map defined as follows. For  $\gamma \in \mathbb{F}_2^E$  and a vertex  $v \in V$ ,

$$(\partial \gamma)_v = \sum_{\substack{e \in E \\ v \in e}} \gamma_e.$$

**Definition** (Definition 1: Coboundary Map). The **coboundary map**  $\delta : \mathbb{F}_2^V \rightarrow \mathbb{F}_2^E$  is the  $\mathbb{F}_2$ -linear map defined as follows. For  $f \in \mathbb{F}_2^V$  and an edge  $e = \{a, b\} \in E$ ,

$$(\delta f)_e = f(a) + f(b).$$

The fundamental relationship between these two maps is that they are transposes of each other with respect to the standard inner product on  $\mathbb{F}_2$ . This duality reflects the geometric fact that boundary and coboundary operations are adjoint to each other.

**Theorem** (Theorem 1: Coboundary Map Is Transpose of Boundary Map). *The coboundary map  $\delta$  is the transpose of the boundary map  $\partial$  with respect to the standard  $\mathbb{F}_2$  inner product. That is, for all  $f \in \mathbb{F}_2^V$  and  $\gamma \in \mathbb{F}_2^E$ ,*

$$\sum_{e \in E} (\delta f)_e \cdot \gamma_e = \sum_{v \in V} f_v \cdot (\partial \gamma)_v.$$

*Proof.* We expand both sides using the definitions of  $\delta$  and  $\partial$ . The left-hand side becomes:

$$\text{LHS} = \sum_{e \in E} (\delta f)_e \cdot \gamma_e = \sum_{e \in E} (f(a) + f(b)) \cdot \gamma_e$$

where  $e = \{a, b\}$ . The right-hand side becomes:

$$\text{RHS} = \sum_{v \in V} f_v \cdot (\partial \gamma)_v = \sum_{v \in V} f_v \cdot \sum_{\substack{e \in E \\ v \in e}} \gamma_e = \sum_{v \in V} \sum_{\substack{e \in E \\ v \in e}} f_v \cdot \gamma_e.$$

We can rewrite the right-hand side by exchanging the order of summation:

$$\text{RHS} = \sum_{e \in E} \sum_{v \in e} f_v \cdot \gamma_e.$$

For each edge  $e = \{a, b\}$  with  $a \neq b$  (since  $G$  is simple), the inner sum  $\sum_{v \in e} f_v$  equals  $f_a + f_b = f(a) + f(b)$ . Therefore:

$$\text{RHS} = \sum_{e \in E} (f(a) + f(b)) \cdot \gamma_e = \text{LHS}.$$

□

For higher-dimensional structures, we extend these concepts to relate edges with 2-dimensional cycles or plaquettes. This gives rise to second boundary and coboundary maps that are essential in topological quantum error correction.

**Definition** (Definition 1: Second Boundary Map). Let  $C$  be a finite type of cycles (or plaquettes), and let each  $c \in C$  be associated with a set of edges  $\text{cycles}(c) \subseteq E$ . The **second boundary map**  $\partial_2 : \mathbb{F}_2^C \rightarrow \mathbb{F}_2^E$  is the  $\mathbb{F}_2$ -linear map defined by: for  $\sigma \in \mathbb{F}_2^C$  and an edge  $e \in E$ ,

$$(\partial_2 \sigma)_e = \sum_{\substack{c \in C \\ e \in \text{cycles}(c)}} \sigma_c.$$

**Definition** (Definition 1: Second Coboundary Map). The **second coboundary map**  $\delta_2 : \mathbb{F}_2^E \rightarrow \mathbb{F}_2^C$  is the  $\mathbb{F}_2$ -linear map defined by: for  $\gamma \in \mathbb{F}_2^E$  and a cycle  $c \in C$ ,

$$(\delta_2 \gamma)_c = \sum_{\substack{e \in E \\ e \in \text{cycles}(c)}} \gamma_e.$$

**Theorem** (Theorem 1: Second Coboundary Map Is Transpose of Second Boundary Map). *The second coboundary map  $\delta_2$  is the transpose of the second boundary map  $\partial_2$ . That is, for all  $\gamma \in \mathbb{F}_2^E$  and  $\sigma \in \mathbb{F}_2^C$ ,*

$$\sum_{c \in C} (\delta_2 \gamma)_c \cdot \sigma_c = \sum_{e \in E} \gamma_e \cdot (\partial_2 \sigma)_e.$$

*Proof.* We expand both sides using the definitions. The left-hand side becomes:

$$\text{LHS} = \sum_{c \in C} \left( \sum_{e \in \text{cycles}(c)} \gamma_e \right) \cdot \sigma_c = \sum_{c \in C} \sum_{e \in E} [e \in \text{cycles}(c)] \gamma_e \sigma_c.$$

The right-hand side becomes:

$$\text{RHS} = \sum_{e \in E} \gamma_e \cdot \left( \sum_{c: e \in \text{cycles}(c)} \sigma_c \right) = \sum_{e \in E} \sum_{c \in C} [e \in \text{cycles}(c)] \gamma_e \sigma_c.$$

Exchanging the order of summation on the left-hand side gives:

$$\text{LHS} = \sum_{e \in E} \sum_{c \in C} [e \in \text{cycles}(c)] \gamma_e \sigma_c = \text{RHS}.$$

□

These boundary and coboundary maps satisfy natural properties when applied to indicator functions, which form a basis for the respective vector spaces. These properties make explicit the incidence relationships encoded in the abstract linear maps.

**Theorem** (Theorem 1: Boundary and Coboundary on Indicators). *The following formulas hold for indicator functions:*

1. For  $e \in E$  and  $v \in V$ :  $(\partial \mathbf{1}_e)(v) = \begin{cases} 1 & \text{if } v \in e \\ 0 & \text{otherwise} \end{cases}$
2. For  $v \in V$  and  $e \in E$ :  $(\delta \mathbf{1}_v)(e) = \begin{cases} 1 & \text{if } v \in e \\ 0 & \text{otherwise} \end{cases}$
3. For  $c \in C$  and  $e \in E$ :  $(\partial_2 \mathbf{1}_c)(e) = \begin{cases} 1 & \text{if } e \in \text{cycles}(c) \\ 0 & \text{otherwise} \end{cases}$
4. For  $e \in E$  and  $c \in C$ :  $(\delta_2 \mathbf{1}_e)(c) = \begin{cases} 1 & \text{if } e \in \text{cycles}(c) \\ 0 & \text{otherwise} \end{cases}$

These maps form the foundation for constructing chain complexes  $\mathbb{F}_2^C \xrightarrow{\partial_2} \mathbb{F}_2^E \xrightarrow{\partial} \mathbb{F}_2^V$  and cochain complexes  $\mathbb{F}_2^V \xrightarrow{\delta} \mathbb{F}_2^E \xrightarrow{\delta_2} \mathbb{F}_2^C$ , whose homology and cohomology groups capture the topological invariants of the underlying graph or surface.

## 1.6 Remark 5: ExactnessOfSequences

The study of chain complexes on graphs provides a fundamental framework for understanding topological properties through algebraic structures. When we have a graph with a collection of cycles, we can construct boundary and coboundary maps that encode the incidence relationships between vertices, edges, and cycles. A key property of these maps is that they satisfy the chain complex condition, meaning that composing consecutive maps in the sequence yields zero.

In this section, we establish the exactness properties of the sequences formed by these boundary and coboundary maps. The central insight is that when cycles satisfy a natural evenness condition (each vertex is incident to an even number of edges of each cycle), the resulting maps form chain complexes. However, for connected graphs, these sequences are not short exact due to the presence of constant functions in the kernel of the coboundary map.

*Remark* (Remark 5: Exactness of Sequences). For a graph  $G = (V, E)$  with a collection of cycles  $C$  (where each cycle  $c \in C$  is specified by its edge set, and each vertex of  $G$  that appears in  $c$  is incident to exactly two edges of  $c$ ), the maps  $\partial_2$  and  $\partial$  satisfy the chain complex property  $\partial \circ \partial_2 = 0$ . Similarly,  $\delta_2 \circ \delta = 0$  holds by transposition. For a connected graph,  $\ker(\delta) = \{0, \mathbf{1}\}$  (constant functions), so the sequences are not short exact since  $\ker(\delta) \neq 0$ . If  $C$  generates the cycle space ( $\text{im}(\partial_2) = \ker(\partial)$ ), then  $\ker(\partial) = \text{im}(\partial_2)$ .

The following sequence of theorems establishes these exactness properties systematically, beginning with the fundamental result that the boundary of any cycle is zero.

**Theorem** (Boundary of a Cycle is Zero). *Let  $G = (V, E)$  be a graph and let  $C$  be a collection of cycles specified by their edge sets. Suppose that for every cycle  $c \in C$  and every vertex  $v \in V$ , the number of edges of  $c$  incident to  $v$  is even. Then for any cycle  $c \in C$  and any vertex  $v \in V$ ,*

$$\partial(\mathbf{1}_c)(v) = 0,$$

where  $\mathbf{1}_c$  is the indicator function of the edges of  $c$ .

*Proof.* By the definition of the boundary map,  $\partial(\mathbf{1}_c)(v) = \sum_{e \in E} \llbracket v \in e \rrbracket \cdot \mathbf{1}_c(e)$ . We rewrite each summand: the term  $\llbracket v \in e \rrbracket \cdot \mathbf{1}_c(e)$  equals  $\llbracket e \in c \wedge v \in e \rrbracket$ , by splitting on cases (if  $v \in e$ , then the value is  $\mathbf{1}_c(e)$ ; otherwise 0, and if  $e \notin c$ , both expressions are 0).

After this rewriting, we apply the identity  $\sum_e \llbracket P(e) \rrbracket = |\{e : P(e)\}|$  (sum of boolean indicators equals the cardinality of the filtered set). This gives  $\partial(\mathbf{1}_c)(v) = |\{e \in E : e \in c \wedge v \in e\}| \pmod{2}$ . Since by hypothesis this cardinality is even, the natural number cast to  $\mathbb{Z}/2\mathbb{Z}$  is zero, as required.  $\square$

**Theorem** (Boundary of Second Boundary on a Single Cycle). *Under the same cycle evenness hypothesis, for any cycle  $c \in C$  and vertex  $v \in V$ ,*

$$\partial(\partial_2(\mathbf{e}_c))(v) = 0,$$

where  $\mathbf{e}_c = \pi_c(1)$  is the standard basis vector corresponding to  $c$ .

*Proof.* We expand both maps using their definitions. By the definition of  $\partial_2$ , for each edge  $e$ ,  $(\partial_2(\mathbf{e}_c))(e) = \sum_{c'} \llbracket e \in c' \rrbracket \cdot \mathbf{e}_c(c')$ . We establish the key identity: for each edge  $e$ ,  $\sum_{c'} \llbracket e \in c' \rrbracket \cdot \mathbf{e}_c(c') = \mathbf{1}_c(e)$ .

To see this, we rewrite the sum by cases: if  $c' = c$  the summand is  $\llbracket e \in c \rrbracket$ , and if  $c' \neq c$  the summand is 0 (since  $\mathbf{e}_c(c') = 0$ ). Thus the sum telescopes via  $\sum_{c'} \llbracket c' = c \rrbracket \cdot \llbracket e \in c' \rrbracket = \llbracket e \in c \rrbracket$  by evaluating at the unique term  $c' = c$ .

Substituting this into the boundary map expression, for each edge  $e$  we replace the inner sum by  $\mathbf{1}_c(e)$  (splitting on whether  $v \in e$  to handle the outer indicator). The resulting expression is exactly  $\partial(\mathbf{1}_c)(v)$ , which equals 0 by the previous theorem.  $\square$

**Theorem** (Chain Complex Property:  $\partial \circ \partial_2 = 0$ ). *Under the cycle evenness hypothesis (every vertex is incident to an even number of edges of each cycle), the composition of the boundary map with the second boundary map is zero:*

$$\partial \circ \partial_2 = 0.$$

*Proof.* By extensionality of linear maps, it suffices to show that for every  $\sigma : C \rightarrow \mathbb{Z}/2\mathbb{Z}$  and every  $v \in V$ ,  $\partial(\partial_2(\sigma))(v) = 0$ .

Expanding the definitions, the left-hand side is

$$\sum_{e \in E} \llbracket v \in e \rrbracket \cdot \left( \sum_{c \in C} \llbracket e \in c \rrbracket \cdot \sigma(c) \right).$$

We rewrite each summand: when  $v \in e$ , the inner expression is  $\sum_c \llbracket e \in c \rrbracket \cdot \sigma(c)$ ; when  $v \notin e$ , the summand is 0. Combining the conditions, each summand becomes  $\sum_c \llbracket e \in c \wedge v \in e \rrbracket \cdot \sigma(c)$ .

Swapping the order of summation, we obtain  $\sum_{c \in C} \sum_{e \in E} \llbracket e \in c \wedge v \in e \rrbracket \cdot \sigma(c)$ . For each fixed  $c$ , we factor out  $\sigma(c)$  to get  $\sigma(c) \cdot \sum_{e \in E} \llbracket e \in c \wedge v \in e \rrbracket$ . The inner sum  $\sum_e \llbracket e \in c \wedge v \in e \rrbracket = |\{e \in E : e \in c \wedge v \in e\}|$  counts edges of  $c$  incident to  $v$ , which is even by hypothesis. Therefore in  $\mathbb{Z}/2\mathbb{Z}$  this count is 0, and each term  $\sigma(c) \cdot 0 = 0$ . Summing over all  $c$  gives 0.  $\square$

The dual results for the coboundary maps follow by similar arguments using the transpose relationship.

**Theorem** (Coboundary Chain Complex Property:  $\delta_2 \circ \delta = 0$ ). *Under the cycle evenness hypothesis, the composition of the second coboundary map with the coboundary map is zero:*

$$\delta_2 \circ \delta = 0.$$

*Proof.* By extensionality, it suffices to show that for every  $f : V \rightarrow \mathbb{Z}/2\mathbb{Z}$  and every  $c \in C$ ,  $\delta_2(\delta(f))(c) = 0$ . Expanding the definitions, the left-hand side equals  $\sum_{e \in E} \llbracket e \in c \rrbracket \cdot (\delta f)(e)$ .

Let  $\mathbf{1}_c : E \rightarrow \mathbb{Z}/2\mathbb{Z}$  be the indicator function of cycle  $c$ . We rewrite the sum as  $\sum_e (\delta f)(e) \cdot \mathbf{1}_c(e)$ . By the transpose identity  $\sum_e (\delta f)(e) \cdot g(e) = \sum_v f(v) \cdot (\partial g)(v)$ , this equals  $\sum_v f(v) \cdot (\partial \mathbf{1}_c)(v)$ .

Since each vertex is incident to an even number of edges of each cycle, we have  $(\partial \mathbf{1}_c)(v) = 0$  for all  $v$ . Therefore the right-hand side becomes  $\sum_v f(v) \cdot 0 = 0$ .  $\square$

The chain complex properties immediately yield the desired inclusion relationships between image and kernel spaces.

**Theorem** ( $\text{im}(\partial_2) \subseteq \ker(\partial)$  and  $\text{im}(\delta) \subseteq \ker(\delta_2)$ ). *Under the cycle evenness hypothesis:*

$$\text{im}(\partial_2) \subseteq \ker(\partial) \quad \text{and} \quad \text{im}(\delta) \subseteq \ker(\delta_2).$$

*Proof.* Both inclusions follow from the characterization that  $\text{im}(A) \subseteq \ker(B)$  if and only if  $B \circ A = 0$ , combined with the chain complex properties established above.  $\square$

For connected graphs, we can completely characterize the kernel of the coboundary map. The key insight is that functions in the kernel must be constant on connected components.

**Definition** (All-Ones Function). The *all-ones function*  $\mathbf{1} : V \rightarrow \mathbb{Z}/2\mathbb{Z}$  is defined by  $\mathbf{1}(v) = 1$  for all  $v \in V$ .

**Theorem** ( $\ker(\delta)$  for Connected Graphs is  $\{0, \mathbf{1}\}$ ). *For a connected graph  $G$ , a function  $f : V \rightarrow \mathbb{Z}/2\mathbb{Z}$  lies in  $\ker(\delta)$  if and only if  $f = 0$  or  $f = \mathbf{1}$ :*

$$f \in \ker(\delta) \iff f = 0 \text{ or } f = \mathbf{1}.$$

*Proof.* We prove both directions.

( $\Rightarrow$ ): Suppose  $f \in \ker(\delta)$ . If  $f(a) = f(b)$  whenever vertices  $a$  and  $b$  are adjacent, then by connectivity,  $f$  must be constant on the entire vertex set. For any edge  $e = \{a, b\}$ , we have  $(\delta f)(e) = f(a) + f(b) = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ , which implies  $f(a) = f(b)$ .

By connectivity, any two vertices are connected by a walk, and we can show by induction on walks that  $f$  takes the same value on all vertices of any walk. Therefore  $f$  is constant. Since  $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$ , either  $f \equiv 0$  or  $f \equiv 1 = \mathbf{1}$ .

( $\Leftarrow$ ): If  $f = 0$ , then  $\delta f = 0$  trivially. If  $f = \mathbf{1}$ , then for any edge  $e = \{a, b\}$ ,  $(\delta f)(e) = f(a) + f(b) = 1 + 1 = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ .  $\square$

This characterization immediately shows that the sequences are not short exact.

**Theorem** (Sequences Are Not Short Exact). *Assuming  $V$  is nonempty, the kernel of the coboundary map is nontrivial:*

$$\ker(\delta) \neq \{0\}.$$

*In particular, the chain complex sequences involving  $\delta$  are not short exact.*

*Proof.* Since  $V$  is nonempty, the all-ones function  $\mathbf{1}$  is well-defined and nonzero. By the previous theorem,  $\mathbf{1} \in \ker(\delta)$ , so the kernel contains a nonzero element.  $\square$

This analysis reveals the fundamental obstruction to short exactness: in a connected graph, there are always nontrivial functions (the constant functions) that lie in the kernel of the coboundary map. The exactness properties depend crucially on whether the cycle collection  $C$  generates the entire cycle space of the graph.

## 1.7 Definition 2: GaussLawAndFluxOperators

The theory of quantum error correction often relies on stabilizer codes, where the code space is defined as the simultaneous eigenspace of a set of commuting Pauli operators called stabilizers. In the context of topological quantum codes built on graphs, two fundamental types of stabilizers emerge naturally from the geometric structure: Gauss's law operators that enforce local constraints at vertices, and flux operators that measure global topological properties around cycles. These operators form the foundation for surface codes and other topological quantum error correcting codes.

**Definition** (Definition 2: Extended Qubit Type and Gauss-Flux Operators). Given a graph  $G = (V, E)$  and a collection of cycles  $C$ , we define operators on an extended qubit system consisting of both vertex and edge qubits.

The **extended qubit type** is  $\text{ExtQubit}(G) := V \oplus G.\text{edgeSet}$ , where vertex qubits are labeled by  $\text{inl}(v)$  and edge qubits by  $\text{inr}(e)$ .

For each vertex  $v \in V$ , the **Gauss's law operator** is

$$A_v = X_v \prod_{e \ni v} X_e,$$

acting with Pauli  $X$  on vertex qubit  $v$  and all incident edge qubits.

For each cycle  $p \in C$ , the **flux operator** is

$$B_p = \prod_{e \in p} Z_e,$$

acting with Pauli  $Z$  on all edge qubits in cycle  $p$ .

The **logical operator** is

$$L = \prod_{v \in V} X_v,$$

acting with Pauli  $X$  on all vertex qubits.

These operators exhibit a rich commutation structure that makes them suitable as stabilizers for quantum error correction. The Gauss operators are pure  $X$ -type and mutually commute since they have disjoint  $Z$ -support (which is empty). Similarly, flux operators are pure  $Z$ -type and mutually commute. The key insight is that Gauss and flux operators also commute with each other, provided the graph satisfies a natural geometric constraint: every cycle must intersect each vertex in an even number of edges (typically 0 or 2). This condition ensures that the symplectic inner product between any Gauss operator and any flux operator vanishes modulo 2.

### 1.8 Definition 3: DeformedOperator

In quantum error correction with gauge fields, it becomes necessary to lift Pauli operators from the original system to an extended system that includes auxiliary degrees of freedom. The deformed operator construction provides a systematic way to perform this lifting while preserving essential commutation relations with stabilizer constraints.

**Definition** (Definition 3: Z-Support on Vertices). The  $Z$ -support on vertices of a Pauli operator  $P$  on  $V$  is the binary vector  $\text{zSupportOnVertices}(P) \in (\mathbb{Z}/2\mathbb{Z})^V$  defined by

$$\text{zSupportOnVertices}(P)(v) = \begin{cases} 1 & \text{if } P.\text{zVec}(v) \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

This is the characteristic function of  $S_Z(P) \cap V$ .

**Definition** (Definition 3: Commutes With Logical). A Pauli operator  $P$  on  $V$  commutes with the logical operator  $L = \prod_{v \in V} X_v$  if the sum of its  $Z$ -support on vertices vanishes in  $\mathbb{Z}/2\mathbb{Z}$ :

$$\sum_{v \in V} \text{zSupportOnVertices}(P)(v) = 0.$$

Equivalently,  $P$  has an even number of vertices with  $Z$ -action.

**Definition** (Definition 3: Boundary Condition). The *boundary condition* for a Pauli operator  $P$  on  $V$  and an edge-path  $\gamma \in (\mathbb{Z}/2\mathbb{Z})^E$  asserts that

$$\partial\gamma = \text{zSupportOnVertices}(P),$$

where  $\partial$  denotes the boundary map of the graph  $G$ .

**Definition** (Definition 3: Deformed Operator). The *deformed operator*  $\tilde{P}$  on the extended qubit system  $V \oplus E$  is defined as follows. Given a Pauli operator  $P$  on  $V$  and an edge-path  $\gamma \in (\mathbb{Z}/2\mathbb{Z})^E$ :

- On vertex qubits ( $v \in V$ ):  $\tilde{P}.\text{xVec}(v) = P.\text{xVec}(v)$  and  $\tilde{P}.\text{zVec}(v) = P.\text{zVec}(v)$ .
- On edge qubits ( $e \in E$ ):  $\tilde{P}.\text{xVec}(e) = 0$  and  $\tilde{P}.\text{zVec}(e) = \gamma(e)$ .

That is,  $\tilde{P}$  acts as  $P$  on vertex qubits and as  $Z_e$  (if  $\gamma(e) = 1$ ) or identity (if  $\gamma(e) = 0$ ) on edge qubits.

**Theorem** (Theorem 3: Deforming the Identity). *Deforming the identity operator with edge-path  $\gamma$  gives a pure- $Z$  edge operator:*

$$\begin{aligned} \tilde{\mathbf{I}}(\gamma).\text{xVec}(q) &= 0 \quad \text{for all } q \in V \oplus E, \\ \tilde{\mathbf{I}}(\gamma).\text{zVec}(q) &= \begin{cases} 0 & \text{if } q \in V, \\ \gamma(e) & \text{if } q = e \in E. \end{cases} \end{aligned}$$

*Proof.* By extensionality, it suffices to check each component. For each qubit  $q \in V \oplus E$ , we case-split on whether  $q$  is a vertex or an edge qubit and simplify using the definition of the deformed operator.  $\square$

**Theorem** (Theorem 3: Self-Inverse Property). *The deformed operator is self-inverse:  $\tilde{P} \cdot \tilde{P} = \mathbf{1}$ .*

*Proof.* By extensionality, it suffices to verify equality on each qubit  $q \in V \oplus E$ .

For the  $\text{xVec}$  component: if  $q = v \in V$ , then  $(\tilde{P} \cdot \tilde{P}).\text{xVec}(v) = P.\text{xVec}(v) + P.\text{xVec}(v) = 0$  by the characteristic-two identity  $a + a = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ . If  $q = e \in E$ , then  $(\tilde{P} \cdot \tilde{P}).\text{xVec}(e) = 0 + 0 = 0$ , which equals  $\mathbf{1}.\text{xVec}(e)$ .

For the  $\text{zVec}$  component: if  $q = v \in V$ , then  $(\tilde{P} \cdot \tilde{P}).\text{zVec}(v) = P.\text{zVec}(v) + P.\text{zVec}(v) = 0$  by the same characteristic-two identity. If  $q = e \in E$ , then  $(\tilde{P} \cdot \tilde{P}).\text{zVec}(e) = \gamma(e) + \gamma(e) = 0$ .  $\square$

**Lemma** (Lemma 3: Sum of Z-Support Equals Cardinality). *For any Pauli operator  $P$  on  $V$ ,*

$$\sum_{v \in V} \text{zSupportOnVertices}(P)(v) = |\{v \in V \mid P.\text{zVec}(v) \neq 0\}| \pmod{2}.$$

*Proof.* Expanding the definition of  $\text{zSupportOnVertices}$ , the left-hand side is  $\sum_{v \in V} \mathbf{1}_{P.\text{zVec}(v) \neq 0}$ , which by rewriting in terms of the boolean indicator function equals the cardinality of  $\{v \in V \mid P.\text{zVec}(v) \neq 0\}$  cast to  $\mathbb{Z}/2\mathbb{Z}$ .  $\square$

**Theorem** (Theorem 3: Commutativity Iff Even Z-Support). *A Pauli operator  $P$  commutes with the logical operator if and only if  $|\{v \in V \mid P.\text{zVec}(v) \neq 0\}|$  is even:*

$$\text{CommutatesWithLogical}(P) \iff 2 \mid |\{v \in V \mid P.\text{zVec}(v) \neq 0\}|.$$

*Proof.* Unfolding the definition of  $\text{CommutatesWithLogical}$ , the condition becomes  $\sum_v \text{zSupportOnVertices}(P)(v) = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ . By Lemma 3, this sum equals the cardinality cast to  $\mathbb{Z}/2\mathbb{Z}$ . The result then follows from the fact that  $n = 0$  in  $\mathbb{Z}/2\mathbb{Z}$  if and only if  $n$  is even.  $\square$

**Theorem** (Theorem 3: Boundary Sum Equals Zero). *For any edge-path  $\gamma \in (\mathbb{Z}/2\mathbb{Z})^E$ ,*

$$\sum_{v \in V} (\partial\gamma)(v) = 0,$$

*since each edge contributes to exactly two vertices.*

*Proof.* Expanding the definition of the boundary map, we have

$$\sum_{v \in V} (\partial\gamma)(v) = \sum_{v \in V} \sum_{e \in E} \begin{cases} \gamma(e) & \text{if } v \in e, \\ 0 & \text{otherwise.} \end{cases}$$

Swapping the order of summation, it suffices to show that each inner sum  $\sum_{v \in V} \mathbf{1}_{v \in e} \cdot \gamma(e)$  vanishes. We factor out  $\gamma(e)$  to obtain  $(\sum_{v \in V} \mathbf{1}_{v \in e}) \cdot \gamma(e)$ .

For each edge  $e = \{a, b\}$  with  $a \neq b$  (since the graph has no loops), we have  $\sum_{v \in V} \mathbf{1}_{v \in \{a, b\}} = \mathbf{1}_{v=a} + \mathbf{1}_{v=b}$ . After evaluating the sum over all vertices, we get the contribution  $1 + 1 = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ , so each edge's contribution is  $0 \cdot \gamma(e) = 0$ . The total sum is therefore 0.  $\square$

**Theorem** (Theorem 3: Boundary Condition Implies Commutativity). *If the boundary condition  $\partial\gamma = \text{zSupportOnVertices}(P)$  holds, then  $P$  commutes with the logical operator.*

*Proof.* Unfolding the definition of `CommutatesWithLogical`, we need to show  $\sum_v \text{zSupportOnVertices}(P)(v) = 0$ . Unfolding the boundary condition  $\partial\gamma = \text{zSupportOnVertices}(P)$ , we rewrite the goal as  $\sum_v (\partial\gamma)(v) = 0$ . This follows directly from the preceding theorem.  $\square$

**Theorem** (Theorem 3: Commutation with Gauss's Law). *Let  $P$  be a Pauli operator on  $V$ ,  $\gamma$  an edge-path, and suppose the boundary condition  $\partial\gamma = \text{zSupportOnVertices}(P)$  holds. Then for every vertex  $v \in V$ , the deformed operator  $\tilde{P}$  commutes with the Gauss's law operator  $A_v$ :*

$$\text{PauliCommute}(\tilde{P}, A_v).$$

*Proof.* Expanding the definition of `PauliCommute` and `symplecticInner`, we need to show

$$\sum_{q \in V \oplus E} (\tilde{P}.x\text{Vec}(q) \cdot A_v.z\text{Vec}(q) + \tilde{P}.z\text{Vec}(q) \cdot A_v.x\text{Vec}(q)) = 0.$$

We split the sum over the type  $V \oplus E$ .

**Vertex contribution:** We first establish that

$$\sum_{w \in V} (\tilde{P}.x\text{Vec}(w) \cdot A_v.z\text{Vec}(w) + \tilde{P}.z\text{Vec}(w) \cdot A_v.x\text{Vec}(w)) = P.z\text{Vec}(v).$$

Expanding the definitions of  $\tilde{P}$  and  $A_v$ , the  $A_v.z\text{Vec}$  component on vertices is 0, so the first term vanishes. The  $A_v.x\text{Vec}$  component on vertex  $w$  is  $\mathbf{1}_{w=v}$ , so the sum reduces to  $\sum_w P.z\text{Vec}(w) \cdot \mathbf{1}_{w=v} = P.z\text{Vec}(v)$ .

**Edge contribution:** We establish that

$$\sum_{e \in E} (\tilde{P}.x\text{Vec}(e) \cdot A_v.z\text{Vec}(e) + \tilde{P}.z\text{Vec}(e) \cdot A_v.x\text{Vec}(e)) = \sum_{e \in E} \begin{cases} \gamma(e) & \text{if } v \in e, \\ 0 & \text{otherwise.} \end{cases}$$

Since  $\tilde{P}.x\text{Vec}(e) = 0$  on edges, the first term vanishes. The second term is  $\gamma(e) \cdot \mathbf{1}_{v \in e}$ , which gives the claimed expression after case-splitting on whether  $v \in e$ .

**Combining:** The total symplectic inner product is  $P.z\text{Vec}(v) + \sum_e \mathbf{1}_{v \in e} \cdot \gamma(e)$ . From the boundary condition,  $(\partial\gamma)(v) = \text{zSupportOnVertices}(P)(v)$ . Expanding the boundary map,  $(\partial\gamma)(v) = \sum_e \mathbf{1}_{v \in e} \cdot \gamma(e)$ . Rewriting using the boundary condition, the total becomes  $P.z\text{Vec}(v) + \text{zSupportOnVertices}(P)(v)$ . If  $P.z\text{Vec}(v) = 0$ , then  $\text{zSupportOnVertices}(P)(v) = 0$  and the sum is 0. If  $P.z\text{Vec}(v) \neq 0$ , then  $P.z\text{Vec}(v) = 1$  and  $\text{zSupportOnVertices}(P)(v) = 1$  in  $\mathbb{Z}/2\mathbb{Z}$ , so the sum is  $1 + 1 = 0$  by the characteristic-two identity.  $\square$

**Theorem** (Theorem 3: Compatibility with Multiplication). *For Pauli operators  $P, Q$  on  $V$  and edge-paths  $\gamma_1, \gamma_2$ ,*

$$\tilde{P}(\gamma_1) \cdot \tilde{Q}(\gamma_2) = \widetilde{P \cdot Q}(\gamma_1 + \gamma_2).$$

*Proof.* By extensionality, we verify equality on each qubit  $q \in V \oplus E$ .

For the `xVec` component: if  $q = v \in V$ , then both sides equal  $P.x\text{Vec}(v) + Q.x\text{Vec}(v)$  by simplification using the definition of the deformed operator. If  $q = e \in E$ , then both sides equal  $0 + 0 = 0$ .

For the `zVec` component: if  $q = v \in V$ , then both sides equal  $P.z\text{Vec}(v) + Q.z\text{Vec}(v)$ . If  $q = e \in E$ , then both sides equal  $\gamma_1(e) + \gamma_2(e)$  by the pointwise addition of functions.  $\square$

The deformed operator construction provides a natural way to extend Pauli operators to the gauge-theoretic setting while preserving the algebraic structure and ensuring compatibility with gauge constraints. The key insight is that the boundary condition precisely captures the obstruction to commutativity with the logical operator, transforming it into a topological constraint on edge paths.

### 1.9 Remark 6: Noncommuting Operators Cannot Be Deformed

The notion of deformation in quantum error correction refers to the possibility of modifying a Pauli operator by multiplying with additional operators to achieve desired commutation properties. A natural question arises: can we always deform a noncommuting operator to make it commute with the logical space? The answer is negative, as demonstrated by topological obstructions in the underlying graph structure.

*Remark (Remark 6: Noncommuting Operators Cannot Be Deformed).* A Pauli operator  $P$  that does not commute with the logical operator  $L$  cannot be deformed to commute with all Gauss's law operators  $A_v$ . The boundary condition  $\partial\gamma = S_Z(P)|_V$  has no solution when  $\text{CommutatesWithLogical}(P)$  fails, and multiplying  $P$  by  $Z_e$  operators or commuting stabilizers cannot remedy this fundamental obstruction.

The key insight is that the commutation properties are controlled by topological invariants that cannot be changed by local modifications. This leads to several important structural results about the impossibility of deformation.

**Theorem (Theorem 1: Noncommuting Operators Cannot Be Deformed).** *If  $\neg \text{CommutatesWithLogical}(P)$ , then there is no edge-path  $\gamma: E \rightarrow \mathbb{Z}/2\mathbb{Z}$  satisfying the boundary condition  $\text{BoundaryCondition}(G, P, \gamma)$ .*

*Proof.* We proceed by contradiction. Suppose there exists  $\gamma$  with  $\text{BoundaryCondition}(G, P, \gamma)$ . Then by the fundamental result that boundary conditions imply commutation, we conclude  $\text{CommutatesWithLogical}(P)$ , contradicting the hypothesis  $\neg \text{CommutatesWithLogical}(P)$ .  $\square$

The  $Z$ -support function exhibits additive behavior under Pauli multiplication, which is crucial for understanding how deformation attempts interact.

**Theorem (Theorem 2:  $Z$ -Support Sum Additivity).** *The sum of  $\text{zSupportOnVertices}$  is additive under Pauli multiplication:*

$$\sum_{v \in V} \text{zSupportOnVertices}(P \cdot Q)(v) = \sum_{v \in V} \text{zSupportOnVertices}(P)(v) + \sum_{v \in V} \text{zSupportOnVertices}(Q)(v).$$

*Proof.* By distributivity of finite sums, we have  $\sum_v f(v) + \sum_v g(v) = \sum_v (f(v) + g(v))$ , so it suffices to show equality pointwise. For each vertex  $v$ , this follows directly from the multiplicativity of  $Z$ -support, which gives  $\text{zSupportOnVertices}(P \cdot Q)(v) = \text{zSupportOnVertices}(P)(v) + \text{zSupportOnVertices}(Q)(v)$ .  $\square$

**Theorem (Theorem 3: Multiplication Preserves  $\text{CommutatesWithLogical}$ ).** *If  $\text{CommutatesWithLogical}(P)$  and  $\text{CommutatesWithLogical}(Q)$ , then  $\text{CommutatesWithLogical}(P \cdot Q)$ .*

*Proof.* Unfolding the definition of  $\text{CommutatesWithLogical}$  in all hypotheses and the goal, we have  $\sum_v \text{zSupportOnVertices}(P)(v) = 0$  and  $\sum_v \text{zSupportOnVertices}(Q)(v) = 0$ . By  $Z$ -support sum additivity,

$$\sum_v \text{zSupportOnVertices}(P \cdot Q)(v) = 0 + 0 = 0.$$

□

**Theorem** (Theorem 4: Contrapositive of Multiplication Preservation). *If  $\neg$  CommutesWithLogical( $P \cdot Q$ ) and CommutesWithLogical( $Q$ ), then  $\neg$  CommutesWithLogical( $P$ ).*

*Proof.* Assume for contradiction that CommutesWithLogical( $P$ ) holds. Then by Theorem 3 applied to  $P$  and  $Q$ , we obtain CommutesWithLogical( $P \cdot Q$ ), contradicting the hypothesis. □

Single-vertex Pauli-Z operators provide concrete examples of noncommuting operators with specific Z-support properties.

**Lemma** (Lemma 1: Z-Support Sum of pauliZ). *The sum of zSupportOnVertices of pauliZ( $v$ ) equals 1:*

$$\sum_{w \in V} \text{zSupportOnVertices}(\text{pauliZ}(v))(w) = 1.$$

*Proof.* Unfolding zSupportOnVertices, the goal becomes showing

$$\sum_w \begin{cases} 1 & \text{if } (\text{pauliZ}(v)).zVec(w) \neq 0 \\ 0 & \text{otherwise} \end{cases} = 1.$$

For each  $w$ , the indicator function equals 1 if  $w = v$  and 0 if  $w \neq v$ , since the Z-vector of pauliZ( $v$ ) is nonzero precisely at vertex  $v$ . The sum becomes  $\sum_w \llbracket w = v \rrbracket = 1$ , as required. □

**Theorem** (Theorem 5: pauliZ Does Not Commute With Logical). *For any vertex  $v$ ,  $\neg$  CommutesWithLogical(pauliZ( $v$ )).*

*Proof.* Unfolding CommutesWithLogical, we need to show  $\sum_w \text{zSupportOnVertices}(\text{pauliZ}(v))(w) \neq 0$ . By Lemma 1, this sum equals 1. Since  $1 \neq 0$  in  $\mathbb{Z}/2\mathbb{Z}$ , the result follows. □

**Theorem** (Theorem 6: pauliZ Flips CommutesWithLogical). *Multiplying by pauliZ( $v$ ) flips the CommutesWithLogical condition:*

$$\text{CommutesWithLogical}(P \cdot \text{pauliZ}(v)) \iff \neg \text{CommutesWithLogical}(P).$$

*Proof.* By Z-support sum additivity and Lemma 1, we have

$$\sum_w \text{zSupportOnVertices}(P \cdot \text{pauliZ}(v))(w) = \sum_w \text{zSupportOnVertices}(P)(w) + 1.$$

The goal reduces to showing  $\sum_v \text{zSupportOnVertices}(P)(v) + 1 = 0 \iff \sum_v \text{zSupportOnVertices}(P)(v) \neq 0$  in  $\mathbb{Z}/2\mathbb{Z}$ .

For the forward direction: if  $\sum_v \text{zSupportOnVertices}(P)(v) + 1 = 0$  and  $\sum_v \text{zSupportOnVertices}(P)(v) = 0$ , then  $0 + 1 = 0$ , giving  $1 = 0$ , a contradiction.

For the reverse direction: if  $\sum_v \text{zSupportOnVertices}(P)(v) \neq 0$ , then in  $\mathbb{Z}/2\mathbb{Z}$  this sum must equal 1. Hence  $1 + 1 = 0$  by the characteristic-two property. □

The formalization extends to the full qubit system  $V \oplus E$ , requiring careful treatment of vertex versus edge qubits.

**Definition** (Definition 1: Z-Support Restricted to Vertices). For a Pauli operator  $P$  on the extended qubit system  $V \oplus E$ , the **Z-support restricted to vertices** is defined as

$$\text{zSupportRestricted}(P) = \sum_{v \in V} \begin{cases} 1 & \text{if } P.zVec(\text{inl}(v)) \neq 0 \\ 0 & \text{otherwise} \end{cases}.$$

This captures the Z-support on the vertex qubits only.

**Definition** (Definition 2: CommutesWithLogical on Extended System). For a Pauli operator  $P$  on the extended qubit system  $V \oplus E$ , we define  $\text{CommutesWithLogical}'(P)$  as the condition that  $\text{zSupportRestricted}(G, P) = 0$ .

**Theorem** (Theorem 7: Edge pauliZ Preserves CommutesWithLogical'). *Multiplying a Pauli operator  $P$  on the extended system  $V \oplus E$  by  $\text{pauliZ}(\text{inr}(e))$  for an edge qubit  $e$  does not change the  $\text{CommutesWithLogical}'$  condition:*

$$\text{CommutesWithLogical}'(P \cdot \text{pauliZ}(\text{inr}(e))) \iff \text{CommutesWithLogical}'(P).$$

*This is because pauliZ on an edge qubit has zero Z-support on vertex qubits.*

*Proof.* It suffices to show that for each vertex  $v$ , we have  $(P \cdot \text{pauliZ}(\text{inr}(e))).\text{zVec}(\text{inl}(v)) = P.\text{zVec}(\text{inl}(v))$ . Since  $\text{inl}(v) \neq \text{inr}(e)$ , we have  $(\text{pauliZ}(\text{inr}(e))).\text{zVec}(\text{inl}(v)) = 0$ . The multiplication formula gives

$$(P \cdot \text{pauliZ}(\text{inr}(e))).\text{zVec}(\text{inl}(v)) = P.\text{zVec}(\text{inl}(v)) + 0 = P.\text{zVec}(\text{inl}(v)).$$

□

The fundamental obstruction persists even when attempting to modify operators with stabilizers.

**Theorem** (Theorem 8: Stabilizer Preserves Non-Commuting Status). *If  $\neg \text{CommutesWithLogical}(P)$  and  $\text{CommutesWithLogical}(s)$ , then  $\neg \text{CommutesWithLogical}(P \cdot s)$ .*

*Proof.* Assume for contradiction that  $\text{CommutesWithLogical}(P \cdot s)$  holds. Since  $P \cdot s \cdot s = P$  (using associativity, Pauli self-inverse property  $s \cdot s = \text{id}$ , and the identity law), we can rewrite  $P = P \cdot s \cdot s$ . By Theorem 3 applied to  $P \cdot s$  and  $s$ , we obtain  $\text{CommutesWithLogical}(P)$ , contradicting the hypothesis. □

**Theorem** (Theorem 9: CommutesWithLogical Invariant Under Commuting Multiplication). *If  $\text{CommutesWithLogical}(Q)$ , then*

$$\text{CommutesWithLogical}(P \cdot Q) \iff \text{CommutesWithLogical}(P).$$

*Proof.* For the forward direction: assume  $\text{CommutesWithLogical}(P \cdot Q)$ . Since  $P \cdot Q \cdot Q = P$ , Theorem 3 applied to  $P \cdot Q$  and  $Q$  gives  $\text{CommutesWithLogical}(P)$ .

For the reverse direction: assume  $\text{CommutesWithLogical}(P)$ . Then Theorem 3 applied to  $P$  and  $Q$  directly gives  $\text{CommutesWithLogical}(P \cdot Q)$ . □

**Theorem** (Theorem 10: No Modification Helps). *If  $\neg \text{CommutesWithLogical}(P)$  and  $\text{CommutesWithLogical}(Q)$ , then  $\neg \text{CommutesWithLogical}(P \cdot Q)$ . No product of  $Z_e$  operators and commuting stabilizers can help.*

*Proof.* By Theorem 9, we have  $\text{CommutesWithLogical}(P \cdot Q) \iff \text{CommutesWithLogical}(P)$ . Since  $\neg \text{CommutesWithLogical}(P)$ , we conclude  $\neg \text{CommutesWithLogical}(P \cdot Q)$ . □

Finally, we establish the connection between boundary conditions and range membership in the boundary map.

**Lemma** (Lemma 2: Boundary Condition Equivalent to Range Membership). *The existence of a boundary condition for  $P$  is equivalent to the  $Z$ -support on vertices being in the image of the boundary map:*

$$(\exists \gamma, \text{BoundaryCondition}(G, P, \gamma)) \iff \text{zSupportOnVertices}(P) \in \text{range}(\partial).$$

*Proof.* Both directions follow directly from the definition of `BoundaryCondition` and range membership. The boundary condition provides exactly the witness needed for range membership, and conversely, range membership provides the required boundary condition.  $\square$

**Theorem** (Theorem 11:  $Z$ -Support Not in Range of Boundary Map). *If  $\neg$  CommutesWithLogical( $P$ ), then  $\text{zSupportOnVertices}(P) \notin \text{range}(\partial)$ .*

*Proof.* By Lemma 2, the goal is equivalent to showing  $\neg \exists \gamma, \text{BoundaryCondition}(G, P, \gamma)$ . This follows directly from Theorem 1.  $\square$

This collection of results establishes that noncommuting operators face fundamental topological obstructions that cannot be overcome by any combination of edge operations or stabilizer multiplication. The  $Z$ -support pattern encodes essential information about the operator's relationship to the logical space, and this information is preserved under all allowed modifications.

## 1.10 Definition 4: DeformedCode

Quantum error correction codes based on lattice gauge theories require a systematic approach to constructing stabilizer generators that respect the underlying gauge symmetry. The deformation procedure extends the original quantum code to include auxiliary edge qubits, creating new stabilizer checks that maintain the physical constraints of gauge theory while preserving the error-correcting properties of the original code.

**Definition** (Definition 4: Deformed Code). The **deformed code** is a quantum error-correcting code defined on an extended qubit system  $V \oplus E(G)$  where  $V$  represents the original vertex qubits and  $E(G)$  represents auxiliary edge qubits. The stabilizer group is generated by three types of operators:

**Gauss's Law Checks:** For each vertex  $v \in V$ ,

$$A_v = \text{gaussLawOp}(G, v)$$

**Flux Checks:** For each cycle  $p \in C$ ,

$$B_p = \text{fluxOp}(G, \text{cycles}, p)$$

**Deformed Original Checks:** For each original check index  $j \in J$ ,

$$\tilde{s}_j = \text{deformedOpExt}(G, s_j, \gamma_j)$$

where  $\gamma_j : E(G) \rightarrow \mathbb{Z}_2$  is an edge-path satisfying the boundary condition  $\partial \gamma_j = S_Z(s_j)|_V$ .

The complete set of stabilizer generators is indexed by the type  $\text{CheckIndex}(V, C, J) = V \oplus C \oplus J$  and given by the function:

$$\begin{aligned} \text{deformedCodeChecks}(\text{gaussLaw}(v)) &= A_v \\ \text{deformedCodeChecks}(\text{flux}(p)) &= B_p \\ \text{deformedCodeChecks}(\text{deformed}(j)) &= \tilde{s}_j \end{aligned}$$

The deformed code construction preserves essential properties of quantum stabilizer codes. All stabilizer generators are Hermitian and self-inverse, following from the general properties of Pauli operators. The Gauss's law checks are pure  $X$ -type operators that implement local gauge constraints at each vertex, while the flux checks are pure  $Z$ -type operators that detect topological flux around cycles. The deformed original checks extend the stabilizers from the original code while respecting the gauge structure through the boundary condition on the edge-paths  $\gamma_j$ .

A crucial property is that all stabilizer generators mutually commute under appropriate conditions. The commutation relations follow systematic patterns: Gauss's law checks always commute among themselves and with deformed checks due to the boundary condition. Flux checks commute among themselves as they are all pure  $Z$ -type. The Gauss-flux commutation requires that cycles have even intersection number with vertices, while deformed-flux commutation follows from the  $X$ - $Z$  orthogonality since deformed checks have no  $X$ -support on edges and flux checks are pure  $Z$ -type. Finally, deformed checks inherit their commutation properties from the original code on vertex qubits, as they have no  $X$ -support on edge qubits.

### 1.11 Statement : QEC1

This is the root import file for the QEC1 library. It serves as an organizational module that aggregates all definitions, theorems, and lemmas by importing the following components:

- **Rem 1:** Notation for Binary Vectors
- **Rem 2:** Notation for Pauli Operators
- **Rem 3:** Notation for Stabilizer Codes
- **Rem 4:** Notation for Cheeger Constant
- **Def 1:** Boundary and Coboundary Maps
- **Rem 5:** Exactness of Sequences
- **Def 2:** Gauss Law and Flux Operators
- **Def 3:** Deformed Operator
- **Rem 6:** Noncommuting Operators Cannot Be Deformed
- **Def 4:** Deformed Code
- **Lem 1:** Deformed Code Checks

The QEC1 library provides a formalization of quantum error-correcting codes using topological methods. The mathematical framework builds on chain complexes over finite fields, with particular emphasis on the relationship between boundary operators, stabilizer codes, and code deformations. This modular structure allows for systematic development of the theory, from basic notation through fundamental definitions to key structural results about deformed codes.

## 1.12 Lemma 1: DeformedCodeChecks

The deformed code construction from the previous chapter introduces a systematic way to extend stabilizer codes by adding new gauge degrees of freedom. A crucial question is whether the resulting operators indeed form a valid stabilizer code. This chapter establishes that all the operators defined in the deformed code construction—Gauss’s law operators  $A_v$ , flux operators  $B_p$ , and deformed checks  $\tilde{s}_j$ —satisfy the fundamental requirements for stabilizer codes: they pairwise commute and are self-inverse.

The key insight is that the deformation process preserves the commutation structure while extending the code to a larger Hilbert space. The Gauss’s law operators act as pure  $X$ -type operators on vertex qubits, while flux operators act as pure  $Z$ -type operators on edge qubits. The deformed original checks inherit their commutation properties from the original stabilizer code while gaining additional  $Z$ -support on edge qubits through the deformation process.

**Lemma** (Lemma 1: Deformed Code Checks Form Valid Stabilizer Code). *The operators from Definition 4 (Deformed Code) form a valid generating set of commuting checks for a stabilizer code. Specifically, all pairwise commutation relations among the Gauss’s law operators  $A_v$ , flux operators  $B_p$ , and deformed checks  $\tilde{s}_j$  hold. The combined checks are self-inverse and together define a valid stabilizer code on the extended qubit system  $V \oplus E(G)$ .*

*Proof.* We establish the result by verifying all required commutation relations and properties.

**Step 1: Gauss’s law operators commute.** For any vertices  $v, w \in V$ , we have  $[A_v, A_w] = 0$ . This follows because both  $A_v$  and  $A_w$  are pure  $X$ -type operators, and distinct  $X$ -type Pauli operators always commute.

**Step 2: Flux operators commute.** For any cycle indices  $p, q \in C$ , we have  $[B_p, B_q] = 0$ . This follows because both  $B_p$  and  $B_q$  are pure  $Z$ -type operators acting on edge qubits, and distinct  $Z$ -type Pauli operators always commute.

**Step 3: Deformed checks commute.** For any indices  $i, j \in J$ , we have  $[\tilde{s}_i, \tilde{s}_j] = 0$ . On edge qubits, both deformed checks are pure  $Z$ -type, so they commute. On vertex qubits, commutation is inherited from the assumption that the original checks  $s_i$  and  $s_j$  commute.

**Step 4: Gauss’s law and flux operators commute.** For any vertex  $v \in V$  and cycle index  $p \in C$ , we have  $[A_v, B_p] = 0$ . The commutator vanishes because the symplectic inner product counts the overlap of the  $X$ -support of  $A_v$  (incident edges to  $v$ ) with the  $Z$ -support of  $B_p$  (edges in cycle  $p$ ). For a valid cycle, the number of edges incident to any vertex is even (0 or 2), ensuring commutation.

**Step 5: Gauss’s law and deformed checks commute.** For any vertex  $v \in V$  and check index  $j \in J$ , we have  $[A_v, \tilde{s}_j] = 0$ . The boundary condition in the deformation process ensures that anticommutation signs from the  $X$ - $Z$  overlaps cancel out appropriately.

**Step 6: Flux and deformed checks commute.** For any cycle index  $p \in C$  and check index  $j \in J$ , we have  $[B_p, \tilde{s}_j] = 0$ . This follows because  $B_p$  is pure  $Z$ -type acting only on edges, while  $\tilde{s}_j$  has no  $X$ -support on edges. Since  $Z$  commutes with  $Z$  and  $B_p$  does not act on vertex qubits where  $\tilde{s}_j$  might have  $X$ -support, they commute.

**Step 7: All operators are self-inverse.** Each operator  $O$  in the set  $\{A_v, B_p, \tilde{s}_j\}$  satisfies  $O^2 = I$ , as required for stabilizer generators.

**Step 8: Valid stabilizer code structure.** The combined set of all checks has index type  $\text{CheckIndex}(V, C, J) = V \oplus C \oplus J$ , and the pairwise commutation established in steps 1-6 ensures this forms a valid stabilizer code on the extended qubit system  $V \oplus E(G)$ .  $\square$

This result is fundamental because it guarantees that the deformation construction preserves the stabilizer structure while extending the code space. The total number of physical qubits becomes

$|V| + |E(G)|$ , and the number of stabilizer checks is  $|V| + |C| + |J|$ . The construction thus provides a systematic method for embedding quantum error-correcting codes into larger systems with additional gauge symmetries, which is essential for fault-tolerant quantum computation schemes that rely on gauge fixing procedures.

### 1.13 Remark 7: Codespace Dimension After Gauging

When studying quantum error correction through the lens of gauge theory, a fundamental question arises: how does the gauging procedure affect the parameters of the resulting quantum code? The gauging process, which transforms a stabilizer code by introducing auxiliary qubits and measurements associated with a graph structure, necessarily alters both the number of physical qubits and the number of stabilizer generators. Understanding this transformation is crucial for determining whether the gauged code retains sufficient logical information to be useful.

The key insight is that gauging consumes exactly one logical qubit from the original code. This consumption occurs because the gauging procedure introduces additional constraints that reduce the dimensionality of the code space in a predictable way. When we start with a stabilizer code having parameters  $[[n, k, d]]$  and apply gauging with respect to a connected graph, the resulting deformed code has parameters  $[[n', k - 1, d']]$  where  $n'$  depends on the graph structure.

*Remark (Remark 7: Codespace Dimension After Gauging).* Consider a stabilizer code acting on  $n$  qubits with  $J$  pairwise-commuting Pauli checks, giving  $k = n - |J|$  logical qubits. When this code is gauged using a connected graph  $G = (V, E)$  with a complete cycle basis  $C$ , the deformed code acts on  $n + |E|$  physical qubits and has  $|V| + |C| + |J|$  stabilizer checks. The key relationship is that  $|C| = |E| - |V| + 1$  for a connected graph, which ensures that the gauging process consumes exactly one logical qubit, yielding  $k_{\text{new}} = k - 1$ .

**Theorem (Deformed Code Number of Qubits).** *Under the gauging construction, the deformed stabilizer code acts on exactly  $|V| + |E|$  physical qubits, where  $|V|$  qubits correspond to the original vertices and  $|E|$  qubits are introduced for the edges of the gauging graph.*

*Proof.* The construction of the deformed code places one qubit at each vertex of the original lattice (contributing  $|V|$  qubits) and introduces one auxiliary qubit for each edge of the gauging graph (contributing  $|E|$  qubits). The total count is therefore  $|V| + |E|$ .  $\square$

**Theorem (Deformed Code Number of Checks).** *The deformed stabilizer code has exactly  $|V| + |C| + |J|$  stabilizer generators, where  $|V|$  generators arise from vertex constraints,  $|C|$  from cycle constraints, and  $|J|$  from the original stabilizer checks.*

*Proof.* The gauging procedure introduces one stabilizer generator for each vertex of the graph (vertex stabilizers), one generator for each cycle in the chosen cycle basis (cycle stabilizers), and preserves all original stabilizer checks. This gives a total of  $|V| + |C| + |J|$  independent stabilizer generators.  $\square$

**Definition (Cycle Rank Property).** For a connected graph  $G = (V, E)$  with cycle collection  $C$ , the **cycle rank property** is the identity

$$|C| + |V| = |E| + 1.$$

This property holds when  $C$  forms a complete cycle basis for the graph, in which case  $|C|$  equals the cycle rank (first Betti number)  $|E| - |V| + 1$ .

**Theorem** (Codespace Dimension Change After Gauging). *Let the original stabilizer code have  $n = |V|$  qubits,  $|J|$  checks, and  $k = n - |J| \geq 1$  logical qubits. If the gauging graph  $G = (V, E)$  with cycle collection  $C$  satisfies the cycle rank property, then the deformed code satisfies*

$$\text{numQubits}(\text{deformed}) - \text{numChecks}(\text{deformed}) = k - 1.$$

*Thus, gauging consumes exactly one logical qubit.*

*Proof.* From the previous theorems, we have  $\text{numQubits}(\text{deformed}) = |V| + |E|$  and  $\text{numChecks}(\text{deformed}) = |V| + |C| + |J|$ . The cycle rank property gives us  $|C| + |V| = |E| + 1$ , which implies  $|E| = |C| + |V| - 1$ .

Substituting this into the difference:

$$\text{numQubits}(\text{deformed}) - \text{numChecks}(\text{deformed}) \tag{1}$$

$$= (|V| + |E|) - (|V| + |C| + |J|) \tag{2}$$

$$= |V| + (|C| + |V| - 1) - |V| - |C| - |J| \tag{3}$$

$$= |V| - 1 - |J| \tag{4}$$

$$= (|V| - |J|) - 1 \tag{5}$$

$$= k - 1. \tag{6}$$

□

The significance of this result extends beyond mere parameter counting. It reveals that the gauging procedure has a uniform effect on the logical dimension regardless of the specific structure of the original code or the complexity of the gauging graph. This one-qubit consumption represents the "cost" of implementing the gauge constraints, and it occurs because the cycle basis introduces exactly one more constraint than the number of new degrees of freedom provided by the edge qubits. This fundamental trade-off between gauge freedom and logical capacity is a central theme in the theory of gauged quantum codes.

### 1.14 Remark 8: FreedomInDeformedChecks

The deformed surface codes described in this work possess a unique codespace, but exhibit significant freedom in the choice of generating sets for their stabilizer generators. This freedom manifests in three distinct ways: the Gauss's law checks  $A_v$  are uniquely determined by the graph structure, the flux checks  $B_p$  can utilize any generating set of cycles for the fundamental group, and most importantly, the deformed checks  $\tilde{s}_j$  allow freedom in selecting the edge-paths  $\gamma_j$  that satisfy the boundary condition.

This flexibility in path selection is not merely a technical artifact—it reflects a deeper mathematical structure. When two different paths  $\gamma$  and  $\gamma'$  both satisfy the same boundary condition  $\partial\gamma = \partial\gamma' = S_Z(s)|_V$ , the corresponding deformed checks differ by a pure  $Z$ -type operator on the edge qubits. Specifically,  $\tilde{s}(\gamma') = \tilde{s}(\gamma) \cdot B_c$  where  $B_c$  is a product of flux operators. The key insight is that the difference  $\gamma + \gamma'$  necessarily lies in the kernel of the boundary map, which corresponds precisely to the cycle space of the graph.

*Remark* (Remark 8: Freedom in Deformed Checks). Different choices of edge-paths  $\gamma_j$  for the deformed checks yield equivalent quantum error-correcting codes. The freedom in path selection is characterized by the cycle space  $\ker(\partial)$ , and different valid choices are related by pure  $Z$ -type operators on the edge qubits that commute with all stabilizer generators.

The mathematical foundation for this equivalence rests on several key structural results. First, we establish that path differences naturally lie in the kernel of the boundary operator, reflecting the topological constraint that cycles form a vector space under symmetric difference.

**Theorem** (Path Difference in Kernel of Boundary). *Let  $G$  be a simple graph on vertices  $V$ , let  $s$  be a Pauli operator on  $V$ , and let  $\gamma, \gamma' : E(G) \rightarrow \mathbb{Z}/2\mathbb{Z}$  be two edge-paths both satisfying the boundary condition for  $s$ , i.e.,*

$$\partial\gamma = z_V(s), \quad \partial\gamma' = z_V(s).$$

*Then  $\partial(\gamma + \gamma') = 0$ .*

*Proof.* Since the boundary map  $\partial = \text{boundaryMap}(G)$  is linear, we have

$$\partial(\gamma + \gamma') = \partial\gamma + \partial\gamma'.$$

Rewriting using the boundary condition, both  $\partial\gamma$  and  $\partial\gamma'$  equal  $z_V(s)$ , so

$$\partial(\gamma + \gamma') = z_V(s) + z_V(s).$$

By extensionality, for each vertex  $v$  we have  $z_V(s)(v) + z_V(s)(v) = 0$  in  $\mathbb{Z}/2\mathbb{Z}$  (since  $a + a = 0$  in characteristic two). Hence  $\partial(\gamma + \gamma') = 0$ .  $\square$

To analyze the algebraic structure of different path choices, we introduce pure  $Z$ -type operators on the edge qubits that encode the path differences.

**Definition** (Pure- $Z$  Edge Operator). Given a simple graph  $G$  and an edge vector  $\delta : E(G) \rightarrow \mathbb{Z}/2\mathbb{Z}$ , the **pure- $Z$  edge operator** is defined as

$$\text{pureZEdgeOp}(G, \delta) := \text{deformedOpExt}(G, \mathbf{1}, \delta),$$

where  $\mathbf{1}$  denotes the identity Pauli operator on  $V$ . This operator acts as the identity on vertex qubits and applies  $Z$  gates on edge qubits according to  $\delta$ .

These pure  $Z$ -type edge operators possess the essential properties needed to characterize the freedom in deformed checks: they are self-inverse, have no  $X$ -support, and most crucially, they commute with all stabilizer generators when the edge vector represents a cycle.

**Theorem** (Pure- $Z$  Edge Operator Commutes with All Checks). *Let  $\delta : E(G) \rightarrow \mathbb{Z}/2\mathbb{Z}$  satisfy  $\partial\delta = 0$ , and let data be any choice of deformed code data. Then for every check index  $a$ ,*

$$\text{PauliCommute}(\text{pureZEdgeOp}(G, \delta), \text{check}_a).$$

*Proof.* We proceed by cases on the type of stabilizer check:

- **Gauss's law checks:** The pure- $Z$  edge operator commutes with  $A_v$  because the boundary condition  $\partial\delta = 0$  ensures that the operator satisfies the deformed operator extension commutation relations.
- **Flux checks:** Both the pure- $Z$  edge operator and flux operators  $B_p$  are pure  $Z$ -type on the edge qubits, so their symplectic inner product vanishes.
- **Deformed checks:** The pure- $Z$  edge operator has zero  $X$ -support on both vertex and edge qubits, making it commute with all deformed checks by the symplectic form calculation.

□

The central result connecting different path choices shows that deformed checks corresponding to different valid paths differ precisely by a pure  $Z$ -type edge operator whose support encodes the path difference.

**Theorem** (Deformed Check Difference is Pure- $Z$  Edge Operator). *Let  $s$  be a Pauli operator on  $V$  and let  $\gamma, \gamma' : E(G) \rightarrow \mathbb{Z}/2\mathbb{Z}$  be two edge-paths. Then*

$$\tilde{s}(\gamma') = \tilde{s}(\gamma) \cdot \text{pureZEdgeOp}(G, \gamma + \gamma').$$

*Proof.* Unfolding the definitions and applying the multiplication formula for deformed operator extensions:

$$\text{deformedOpExt}(G, s, \gamma) \cdot \text{deformedOpExt}(G, \mathbf{1}, \gamma + \gamma') = \text{deformedOpExt}(G, s \cdot \mathbf{1}, \gamma + (\gamma + \gamma')).$$

Simplifying using  $s \cdot \mathbf{1} = s$  and the characteristic-two identity  $\gamma + (\gamma + \gamma') = \gamma'$  yields the result. □

This theoretical framework has important practical implications for quantum error correction. Since different valid path choices yield codes with the same error-correcting properties, one can select paths to optimize specific implementation constraints—such as minimizing circuit depth, reducing connectivity requirements, or improving fault-tolerance thresholds—without compromising the fundamental error-correction capabilities of the deformed surface code.

### 1.15 Definition 5: Gauging Measurement Algorithm

Quantum error correction protocols often require indirect measurements of logical operators through auxiliary systems. When a logical operator does not commute with stabilizer generators, direct measurement would destroy the quantum information. The gauging protocol addresses this by introducing auxiliary edge qubits and measuring carefully chosen operators that collectively reveal the logical operator's eigenvalue while preserving the encoded state.

The gauging measurement algorithm provides a systematic procedure for measuring a logical operator  $L = \prod_{v \in V} X_v$  supported on a connected subgraph. The key insight is that by measuring Gauss's law operators  $A_v$  at each vertex and  $Z$ -operators on auxiliary edge qubits, we can extract the logical measurement result through classical post-processing involving walk parities on the graph.

**Definition** (Definition 5: Gauging Measurement Algorithm). The **gauging measurement algorithm** consists of the following components:

**Input:** A connected graph  $G = (V, E)$  supporting the logical operator  $L = \prod_{v \in V} X_v$ , with a designated base vertex  $v_0 \in V$ .

**Measurement Phase:**

- Measure Gauss's law operators  $A_v = X_v \prod_{e \ni v} X_e$  for each  $v \in V$ , obtaining outcomes  $\varepsilon_v \in \mathbb{Z}/2\mathbb{Z}$
- Measure edge  $Z$ -operators  $Z_e$  for each  $e \in E$ , obtaining outcomes  $\omega_e \in \mathbb{Z}/2\mathbb{Z}$

**Classical Processing:**

- Compute the measurement sign  $\sigma = \sum_{v \in V} \varepsilon_v \in \mathbb{Z}/2\mathbb{Z}$

- For each vertex  $v \in V$ , compute the byproduct correction  $c(v) = \text{walkParity}(\omega, \gamma_v)$  where  $\gamma_v$  is any walk from  $v_0$  to  $v$  and

$$\text{walkParity}(\omega, \gamma) = \sum_{e \in \gamma} \omega_e$$

**Output:** The measurement result  $\sigma$  (where  $\sigma = 0$  indicates the  $+1$  eigenvalue of  $L$  and  $\sigma = 1$  indicates the  $-1$  eigenvalue) and the byproduct correction operator  $\prod_{v:c(v)=1} X_v$ .

The algorithm’s correctness relies on several key mathematical properties. First, all measured operators commute with each other, ensuring measurement order independence. The product of all Gauss’s law operators equals the logical operator  $L$ , which is why  $\sigma$  gives the correct measurement result. The walk parity computation for byproduct corrections is well-defined on connected graphs when all closed walks have zero parity under the edge measurement outcomes.

The use of  $\mathbb{Z}/2\mathbb{Z}$  arithmetic elegantly captures the  $\pm 1$  measurement outcomes:  $0 \leftrightarrow +1$  and  $1 \leftrightarrow -1$ , with products of signs corresponding to sums modulo 2. This encoding simplifies the classical post-processing and makes the commutativity properties manifest through additive structure.

## 1.16 Theorem 1: Gauging Measurement Correctness

Quantum error correction codes must provide fault-tolerant methods for measuring their logical operators. In topological codes, the challenge lies in implementing these measurements while preserving the quantum information encoded in the logical space. The gauging procedure offers a systematic approach to this problem by measuring stabilizer generators (Gauss’s laws) and using the outcomes to perform byproduct corrections that effectively implement a projective measurement of the logical operator.

The mathematical foundation of this procedure relies on the relationship between Gauss’s law operators, the graph structure, and the logical operator itself. When all Gauss’s laws are measured, their product yields the logical operator, but the individual measurement outcomes create byproduct operators that must be corrected to recover the desired projection.

**Theorem** (Theorem 1: Gauging Measurement Correctness). *For a connected graph  $G$ , the gauging measurement procedure implements a projective measurement of the logical operator  $L = \prod_{v \in V} X_v$ . Given an initial code state  $|\psi\rangle$ , the procedure outputs  $(\sigma, |\Psi\rangle)$  where:*

- $\sigma = \sum_v \varepsilon_v \in \mathbb{Z}/2\mathbb{Z}$  encodes the measurement outcome ( $+1$  or  $-1$ ),
- The output state satisfies  $|\Psi\rangle \propto (\mathbf{1} + \sigma L)|\psi\rangle$ , the projection onto the  $\sigma$ -eigenspace of  $L$ .

*Proof.* The proof establishes that the gauging procedure satisfies all requirements for implementing a projective measurement through several key components:

**Step 1: Product Structure.** We first establish that the product of all Gauss’s law operators equals the logical operator:  $\prod_{v \in V} A_v = L$ . This follows directly from the fundamental relationship between stabilizers and logical operators in the quantum error correction code structure.

**Step 2: Byproduct Analysis.** When measuring each Gauss’s law operator  $A_v$ , we obtain outcomes  $\varepsilon_v \in \{0, 1\}$  (corresponding to eigenvalues  $\pm 1$ ). The product of measured operators creates a byproduct correction that depends on the edge measurement outcomes  $\omega$  through the coboundary relationship.

For any binary vector  $c'$  satisfying  $\delta c' = \omega$  (where  $\delta$  is the coboundary map), we can express the effective measurement operator. The key insight is that in a connected graph, the preimage of  $\omega$  under  $\delta$  consists of exactly two elements:  $c'$  and  $c' + \mathbf{1}$  (where  $\mathbf{1}$  is the all-ones vector).

**Step 3: Projector Construction.** The Gauss subset products  $\text{gaussSubsetProduct}(c')$  and  $\text{gaussSubsetProduct}(c' + \mathbf{1})$  satisfy:

$$\text{gaussSubsetProduct}(c' + \mathbf{1}) = \text{gaussSubsetProduct}(c') \cdot L$$

This relationship, combined with the signed outcome additivity  $\varepsilon(c' + \mathbf{1}) = \varepsilon(c') + \sigma$ , ensures that the two terms in the measurement correspond to the identity and logical components of the projector  $\mathbf{1} + \sigma L$ .

**Step 4: Byproduct Correction.** The walk parity vector provides a systematic correction method. Under the closed-walk-zero-parity condition (which holds when edge outcomes satisfy the consistency requirement), the walk parity vector  $c'$  satisfies  $\delta c' = \omega$  and is independent of the choice of walks between vertices.

The byproduct correction operator, constructed from these walk parities, is pure  $X$ -type and commutes with  $L$  restricted to vertex qubits. Applying this correction twice yields the identity, ensuring the correction process is well-defined and reversible.

**Step 5: Measurement Outcome.** The measurement sign  $\sigma = \sum_v \varepsilon_v$  encodes whether the final projection is onto the  $+1$  or  $-1$  eigenspace of  $L$ . The effective operator after all corrections becomes  $(\mathbf{1} + \sigma L)/2$ , which is indeed the projector onto the  $\sigma$ -eigenspace.

**Step 6: Compatibility and Well-definedness.** All individual measurements (Gauss's laws and edge  $Z$  operators) are mutually commuting, ensuring the measurement procedure is well-defined regardless of the order of operations. The correction operator is independent of the choice of spanning tree or walk family used in the classical post-processing.

The combination of these properties establishes that the gauging procedure implements exactly the desired projective measurement of the logical operator  $L$ .  $\square$

This theorem provides the theoretical foundation for fault-tolerant logical measurements in quantum error correction codes. The gauging procedure offers a practical implementation that decomposes the challenging task of measuring  $L$  directly into a sequence of local stabilizer measurements followed by classical post-processing. The resulting protocol preserves the quantum information while providing the measurement outcome, making it suitable for quantum computation where logical measurements are required as intermediate steps rather than final destructive readouts.

### 1.17 Remark 9: CircuitImplementation

In quantum error correction, particularly for codes based on graph structures, the physical implementation of measurement procedures often requires auxiliary qubits and complex multi-step protocols. However, for certain classes of quantum codes, it is possible to perform all necessary measurements using only the data qubits themselves through carefully designed quantum circuits. This approach is particularly elegant for topological codes and gauge theories, where the measurement of gauge constraints can be implemented without expanding the qubit overhead.

The key insight underlying efficient circuit implementation is that controlled-NOT (CNOT) gates can transform stabilizer generators through conjugation, allowing us to measure complex multi-qubit operators by first transforming them into simpler single-qubit measurements. For codes defined on graphs, where stabilizers correspond to local constraints at vertices and edges, this transformation can be achieved systematically using an entangling circuit that couples vertex and edge qubits.

*Remark* (Remark 9: Circuit Implementation). The gauging measurement procedure can be implemented by a quantum circuit using only the original qubits (vertex and edge qubits), without requiring auxiliary ancilla qubits. The protocol consists of six steps:

1. Initialize each edge qubit  $e$  in state  $|0\rangle_e$ .
2. Apply the entangling circuit  $\prod_v \prod_{e \ni v} CX_{v \rightarrow e}$ .
3. Measure  $X_v$  on all vertex qubits  $v \in V$ .
4. Apply the same entangling circuit again.
5. Measure  $Z_e$  on all edge qubits and discard them.
6. Apply byproduct corrections as specified in the measurement protocol.

The effectiveness of this protocol relies on the fact that the entangling circuit transforms the Gauss law operators  $A_v = X_v \prod_{e \ni v} X_e$  into simple Pauli  $X$  measurements: after applying the circuit, measuring  $X_v$  is equivalent to measuring  $A_v$  in the original frame. This transformation occurs because CNOT gates from vertex  $v$  to incident edge  $e$  conjugate  $X_v \mapsto X_v X_e$  while preserving  $X_e \mapsto X_e$ , effectively moving the edge components of  $A_v$  into the vertex measurement.

This circuit-based approach significantly simplifies the practical implementation of gauge theory measurements, reducing both the qubit overhead and the circuit depth compared to ancilla-based protocols. The ability to perform these measurements using only data qubits makes this particularly attractive for near-term quantum devices where qubit count and coherence time are limiting factors.

### 1.18 Remark 10: FlexibilityOfGraphG

Graph choice in quantum error correction is a fundamental design parameter that directly affects code performance. When applying the gauging procedure to transform a stabilizer code, one must select a connected graph  $G$  whose vertex set includes the support of the logical operator. This choice has profound implications for the resulting deformed code's properties.

The flexibility in graph selection arises from two key observations. First, the graph may include additional "dummy" vertices beyond those required by the logical operator's support. Second, different connectivity patterns lead to different numbers of auxiliary edge qubits, affecting both the overhead and the structure of the deformed stabilizer checks.

*Remark* (Remark 10: Flexibility of Graph G). The gauging measurement procedure allows arbitrary choice of the connected graph  $G$  with vertex set equal to (or containing) the support of the logical operator  $L$ . The properties of the deformed code depend strongly on this choice, particularly regarding dummy vertices and graph-theoretic properties that determine edge qubit count, deformed check weights, and code distance.

To formalize the dummy vertex construction, we begin with the standard setup where a logical operator acts on a vertex set  $V$ , then extend it to a larger vertex set  $V \oplus D$  where  $D$  represents additional dummy vertices.

**Definition** (Definition: Original Logical Operator). The original logical operator  $L$  on vertex set  $V$  is defined as

$$L = \prod_{v \in V} X_v,$$

i.e., the Pauli operator with  $x\text{Vec}(v) = 1$  for all  $v \in V$  and  $z\text{Vec} = 0$ .

**Definition** (Definition: Extended Logical Operator). The extended logical operator  $L'$  on the extended vertex type  $V \oplus D$  is defined as

$$L' = \prod_{q \in V \oplus D} X_q,$$

i.e., the Pauli operator with  $\text{xVec}(q) = 1$  for all  $q \in V \oplus D$  and  $\text{zVec} = 0$ .

**Definition** (Definition: Dummy X Product). The dummy  $X$  product  $\prod_{d \in D} X_d$  acting on  $V \oplus D$  is the Pauli operator with

$$\text{xVec}(q) = \begin{cases} 0 & \text{if } q \in V, \\ 1 & \text{if } q \in D, \end{cases} \quad \text{zVec} = 0.$$

**Definition** (Definition: Lifted Logical Operator). The lifted logical operator  $L_{\text{lift}}$  on  $V \oplus D$  acts as  $X$  on all  $V$ -qubits and as the identity on  $D$ -qubits:

$$\text{xVec}(q) = \begin{cases} 1 & \text{if } q \in V, \\ 0 & \text{if } q \in D, \end{cases} \quad \text{zVec} = 0.$$

The key insight is that extending the logical operator to include dummy vertices can be understood as a factorization into the original logical action plus dummy  $X$  operations.

**Theorem** (Theorem: Key Factorization of Extended Logical). *The extended logical operator satisfies*

$$L' = L_{\text{lift}} \cdot \prod_{d \in D} X_d$$

on  $V \oplus D$ .

*Proof.* By extensionality, it suffices to show equality for each component for an arbitrary qubit  $q$ . We consider both the  $\text{xVec}$  and  $\text{zVec}$  components, and case-split on whether  $q \in V$  or  $q \in D$ . In each case, the equality follows by simplification using the definitions of  $L'$ ,  $L_{\text{lift}}$ , and the dummy  $X$  product, together with the formulas for multiplication of Pauli operators.  $\square$

Several basic properties follow immediately from the definitions. The extended logical operator inherits the self-inverse property of Pauli operators.

**Theorem** (Theorem: Extended Logical is Self-Inverse). *The extended logical operator is self-inverse:  $L' \cdot L' = \mathbf{1}$ .*

*Proof.* This follows directly from the fact that every Pauli operator satisfies  $P \cdot P = \mathbf{1}$ .  $\square$

**Theorem** (Theorem: Extended Logical is Pure X). *The extended logical operator  $L'$  is pure X-type: it has no Z-support, i.e.,  $\text{zVec}(L') = 0$ .*

*Proof.* By extensionality over qubits  $q$ , we simplify using the definition of  $L'$ , which has  $\text{zVec} = 0$ .  $\square$

The support structure of the extended logical operator is particularly important for understanding its measurement properties.

**Theorem** (Theorem: X-Support of Extended Logical is Universal). *The X-support of  $L'$  is all of  $V \oplus D$ :  $\text{supportX}(L') = V \oplus D$ .*

*Proof.* By extensionality, for each qubit  $q$ , membership in  $\text{supportX}(L')$  is equivalent to  $\text{xVec}(q) \neq 0$ . By the definition of  $L'$ ,  $\text{xVec}(q) = 1$  for all  $q$ , so every qubit is in the support, yielding  $\text{supportX}(L') = \text{univ}$ .  $\square$

A crucial property for measurement implementation is that dummy vertices with +1 outcomes do not affect the overall logical measurement sign.

**Theorem** (Theorem: Measurement Sign with Dummies). *Let  $\sigma_V : V \rightarrow \mathbb{Z}/2\mathbb{Z}$  be the measurement outcomes on original vertices and  $\sigma_D : D \rightarrow \mathbb{Z}/2\mathbb{Z}$  the outcomes on dummy vertices. If all dummy vertices give +1 outcomes (i.e.,  $\sigma_D(d) = 0$  for all  $d \in D$ ), then*

$$\sum_{q \in V \oplus D} \sigma(q) = \sum_{v \in V} \sigma_V(v).$$

*That is, dummy vertices do not affect the logical measurement sign.*

*Proof.* We rewrite the sum over  $V \oplus D$  as  $\sum_{v \in V} \sigma_V(v) + \sum_{d \in D} \sigma_D(d)$  using the decomposition of a sum over a sum type. Since  $\sigma_D(d) = 0$  for all  $d \in D$  by hypothesis, the second sum vanishes by simplification, yielding the result.  $\square$

The weight of logical operators increases with the number of dummy vertices, which affects the complexity of syndrome extraction.

**Theorem** (Theorem: Weight of Extended Logical). *The weight of the extended logical operator is  $|V| + |D|$ :*

$$\text{weight}(L') = |V| + |D|.$$

*Proof.* Unfolding the definitions of weight and support, and simplifying using the definition of  $L'$  (which has  $\text{xVec}(q) = 1$  for all  $q$  and  $\text{zVec} = 0$ ), the support is all of  $V \oplus D$ . The cardinality of  $V \oplus D$  equals  $|V| + |D|$ .  $\square$

Graph choice affects the deformed code's resource requirements through the number of auxiliary edge qubits introduced by the gauging procedure.

**Theorem** (Theorem: Deformed Code Edge Overhead). *The deformed code's qubit count is  $|V| + |E|$ , so the edge qubit overhead (the additional qubits beyond the original  $|V|$ ) is exactly  $|E(G)|$ . Different graphs  $G$  give different overhead via their edge count:*

$$\text{numQubits}(\tilde{\mathcal{C}}) = |V| + |E(G)|.$$

*Proof.* This follows directly from the established formula for deformed stabilizer code qubit count.  $\square$

The choice of graph also determines the number of stabilizer checks in the deformed code, which affects syndrome extraction complexity.

**Theorem** (Theorem: Deformed Code Check Overhead). *The deformed code's check count is  $|V| + |C| + |J|$ :  $|V|$  Gauss law checks,  $|C|$  flux checks, and  $|J|$  deformed original checks. The overhead in checks beyond the original  $|J|$  is  $|V| + |C|$ :*

$$\text{numChecks}(\tilde{\mathcal{C}}) = |V| + |C| + |J|.$$

*Proof.* This follows directly from the established formula for deformed stabilizer code check count.  $\square$

Graph expansion properties provide a mechanism for controlling the distance of the deformed code through boundary size constraints.

**Theorem** (Theorem: Edge Expansion Lower Bounds Boundary). *For an expander graph  $G$  with expansion constant  $c$ , every nonempty subset  $S \subseteq V$  with  $2|S| \leq |V|$  satisfies*

$$c \cdot |S| \leq |\partial S|,$$

where  $\partial S$  denotes the edge boundary. This is the mechanism by which expansion of  $G$  controls the distance of the deformed code: logical operators supported on small subsets necessarily have large edge boundary.

*Proof.* This follows directly from the established edge boundary cardinality bound for expander graphs, applied to the graph  $G$ , expansion constant  $c$ , the expander hypothesis, subset  $S$ , its nonemptiness, and the size bound  $2|S| \leq |V|$ .  $\square$

The weight structure of deformed checks depends critically on the choice of edge paths, which is determined by the graph topology.

**Theorem** (Theorem: Deformed Check Z-Support on Edges). *The edge Z-support of a deformed check equals the support of the edge-path  $\gamma$ . For any original check  $s$ , edge-path  $\gamma : E(G) \rightarrow \mathbb{Z}/2\mathbb{Z}$ , and edge  $e \in E(G)$ :*

$$\text{zVec}(\tilde{s})(e) = \gamma(e).$$

*More edges in  $\gamma$  means more Z-support on edge qubits, hence higher weight.*

*Proof.* This follows directly from the definition of deformed operator extension, which states that the Z-component of the deformed operator extension on edge qubits equals  $\gamma$ .  $\square$

This flexibility in graph choice represents a fundamental design space in quantum error correction. Sparser graphs minimize auxiliary qubit overhead but may compromise code distance, while denser graphs with good expansion properties can improve distance at the cost of increased resource requirements. The optimal choice depends on the specific constraints and objectives of the quantum computing architecture.

### 1.19 Remark 11: DesiderataForGraphG

When implementing the gauging measurement protocol on quantum LDPC codes, the choice of auxiliary graph  $G$  is crucial for preserving both the code distance and the low-density structure. The graph must balance several competing demands: it needs sufficient connectivity to allow deformation of stabilizer checks while maintaining sparsity to preserve the LDPC property. This leads to three fundamental design criteria that must be satisfied simultaneously.

The first requirement ensures that the deformed stabilizer checks remain local, the second guarantees that the code distance is preserved, and the third maintains the low-weight property of the flux operators that arise in the gauged theory. When these desiderata are met with parameters independent of the system size, the gauging protocol achieves constant overhead per qubit while preserving all essential code properties.

*Remark* (Remark 11: Desiderata for Graph  $G$ ). When choosing a constant-degree graph  $G$  for the gauging measurement, three desiderata must be satisfied:

1. **Short paths for deformation:**  $G$  should contain a constant-length edge-path between any pair of vertices in the  $Z$ -type support of the same original check, ensuring deformed checks have bounded weight.
2. **Sufficient expansion:** The Cheeger constant should satisfy  $h(G) \geq 1$ , ensuring the deformed code has distance at least  $d$ .
3. **Low-weight cycle basis:** There should exist a generating set of cycles where each cycle has weight at most some constant, ensuring the flux operators  $B_p$  have bounded weight.

When all three conditions are satisfied with constants independent of  $n$ , the gauging measurement procedure has constant overhead per qubit, the LDPC property is preserved, and there is no distance reduction.

The interplay between these three requirements captures the essential trade-offs in quantum code construction. The short path condition ensures locality of the deformed stabilizers, the expansion condition provides robustness against errors, and the cycle basis condition maintains computational tractability. Together, they guarantee that the gauged quantum code retains all the beneficial properties of the original LDPC code while enabling the additional measurement capabilities required for the gauging protocol.

## 1.20 Definition 6: CycleSparsifiedGraph

In graph theory and combinatorial optimization, cycles in a graph often contribute to computational complexity when analyzing properties such as sparsity bounds and spectral characteristics. When a graph contains many cycles, particularly those that share edges, direct analysis becomes unwieldy. The cycle-sparsified graph construction addresses this challenge by creating a multilayered structure that distributes cycles across different layers and then applies a systematic triangulation to decompose higher-order cycles into manageable triangular units.

The key insight is that by replicating the original graph across multiple layers and connecting corresponding vertices between adjacent layers, we can assign each cycle from a generating set to a specific layer. This distribution ensures that no single layer bears an excessive cycle load. The triangulation process then breaks down cycles of degree 4 or higher into triangles using a zigzag diagonal pattern, which preserves the essential structural properties while maintaining bounded degree constraints per layer.

**Definition** (Definition 6: Cycle-Sparsified Graph). Let  $G = (V, E)$  be a connected graph with a chosen generating set of cycles  $C$ , and let  $R \in \mathbb{N}$  be the number of additional layers. The **cycle-sparsified graph** (also called the **decongested graph**)  $\overline{\overline{G}}$  with cycle-degree bound  $c$  is constructed as follows:

**Vertex Set:** The sparsified vertex set is  $V \times \text{Fin}(R + 1)$ , where a pair  $(v, i)$  represents vertex  $v$  in layer  $i$ . Layer 0 is designated as the original graph layer.

**Edge Set:** The adjacency relation combines three types of edges:

1. **Intra-layer edges:**  $(p, q)$  are adjacent if  $p_2 = q_2$  and  $G.\text{Adj}(p_1, q_1)$  (same layer, corresponding original vertices adjacent)
2. **Inter-layer edges:**  $(p, q)$  are adjacent if  $p_1 = q_1$  and  $|p_2 - q_2| = 1$  (same original vertex, consecutive layers)

3. **Triangulation edges:**  $(p, q)$  are adjacent if there exists a cycle  $c$  assigned to a non-original layer such that  $p_2 = q_2 = \text{cycleAssignment}(c)$  and  $(p_1, q_1)$  is a zigzag diagonal of  $c$ 's vertex list

**Zigzag Triangulation:** For cycles of length  $m \geq 4$  assigned to layer  $i > 0$ , the zigzag diagonal pattern produces exactly  $m - 2$  diagonals that decompose the  $m$ -gon into triangles.

**Cycle-Degree Bound:** The construction satisfies a per-layer cycle-degree bound  $c$ , meaning that for every edge  $e$  and every layer  $i$ , at most  $c$  cycles assigned to layer  $i$  contain edge  $e$ .

This construction achieves several important properties. First, the total number of vertices in the sparsified graph is  $(R + 1) \cdot |V|$ , representing a controlled expansion. Second, each original edge in  $G$  induces 4-cycles (squares) between consecutive layers, creating a natural "prism" structure. Third, the zigzag triangulation ensures that high-degree cycles are systematically broken down into triangles, which have optimal sparsity properties. Finally, the layer assignment strategy distributes the cycle load, ensuring that the per-layer cycle participation of any edge remains bounded by the parameter  $c$ .

## 1.21 Lemma 2: DecongestionLemmaBound

Cycle-sparsification is a crucial technique in quantum error correction for transforming dense cycle collections into sparse, layered structures. The Freedman-Hastings decongestion lemma provides the fundamental bound on the number of layers required to achieve this sparsification for expander graphs. This result is essential because it ensures that the overhead of sparsification remains polylogarithmic in the graph size, making the technique practical for large quantum codes.

For constant-degree expander graphs with good expansion properties, the decongestion lemma establishes that any collection of cycles can be partitioned into  $O(\log^2 W)$  layers, where each layer has bounded cycle density and  $W$  is the number of vertices. This bound is achieved through a combination of greedy packing strategies and the expansion properties of the underlying graph.

**Lemma (Lemma 2: Decongestion Lemma Bound).** *For a  $\Delta$ -bounded connected expander graph  $G$  on  $W \geq 2$  vertices with Cheeger constant  $h(G) > 0$ , a cycle collection satisfying the cycle rank property  $|C| + |V| = |E| + 1$ , and a positive per-layer bound  $c > 0$ : given an  $M$ -coloring  $f_0 : C \rightarrow \text{Fin}(M + 1)$  with per-layer bound 1 (from König's theorem) where  $M \leq K \cdot \log_2^2(W)$  (from the Freedman-Hastings analysis), there exist  $R$  and  $f$  with*

$$\text{LayerCycleDegreeBound}(f, c) \quad \text{and} \quad R \leq \frac{K \cdot \log_2^2(W)}{c}.$$

*Proof.* The proof proceeds by applying greedy packing techniques to the initial  $M$ -coloring provided by König's theorem.

**Step 1: Initial coloring from König's theorem.** Since the edge-cycle incidence structure forms a bipartite graph between edges and cycles, König's edge coloring theorem guarantees the existence of an  $M$ -coloring  $f_0 : C \rightarrow \text{Fin}(M + 1)$  where each layer (color class) has per-layer bound 1. This means that for any edge  $e$ , at most one cycle in each layer passes through  $e$ .

**Step 2: Greedy packing construction.** Given the initial assignment  $f_0$  and target per-layer bound  $c$ , we define the packed assignment  $f(\gamma) = \lfloor f_0(\gamma)/c \rfloor$ . This maps cycles into  $\text{Fin}(\lfloor M/c \rfloor + 1)$ , so we have  $R \leq \lfloor M/c \rfloor \leq M/c$ .

**Step 3: Verification of per-layer bound.** To show that the packed assignment satisfies the per-layer bound  $c$ , fix an edge  $e$  and a packed layer  $i$ . Let  $S$  be the set of cycles  $\gamma$  with  $f(\gamma) = i$  and  $e \in \gamma$ . We construct an injection  $\varphi : S \rightarrow \text{Fin}(c)$  by  $\varphi(\gamma) = f_0(\gamma) \bmod c$ .

To verify injectivity: suppose  $\gamma_1, \gamma_2 \in S$  with  $f_0(\gamma_1) \bmod c = f_0(\gamma_2) \bmod c$ . Since both are in packed layer  $i$ , we have  $\lfloor f_0(\gamma_1)/c \rfloor = \lfloor f_0(\gamma_2)/c \rfloor$ . By the division algorithm, having equal quotients and equal remainders implies  $f_0(\gamma_1) = f_0(\gamma_2)$ . Since the original assignment  $f_0$  has per-layer bound 1, at most one cycle in the original layer  $f_0(\gamma_1)$  can pass through  $e$ . Therefore  $\gamma_1 = \gamma_2$ .

This establishes that  $|S| \leq |\text{Fin}(c)| = c$ , verifying the per-layer bound.

**Step 4: Application of Freedman-Hastings bound.** The key input is the bound  $M \leq K \cdot \log_2^2(W)$  from the Freedman-Hastings analysis of expander graphs. This bound arises from careful analysis of how cycles can intersect edges in expander graphs with good Cheeger constants.

**Step 5: Final bound.** Combining the packing construction with the expander-specific bound on  $M$ , we obtain

$$R \leq \frac{M}{c} \leq \frac{K \cdot \log_2^2(W)}{c}.$$

□

This lemma is the cornerstone of the sparsification procedure, showing that the number of layers grows only polylogarithmically with the graph size. The dependence on the per-layer bound  $c$  is inversely linear, which allows flexibility in trading off between the number of layers and the sparsity within each layer. The constant  $K$  depends on the degree bound  $\Delta$  and the expansion properties but is independent of the graph size  $W$ , ensuring scalability of the approach.

## 1.22 Remark 12: WorstCaseGraphConstruction

In quantum error correction, constructing graphs that satisfy multiple constraints simultaneously often requires sophisticated techniques. The following construction demonstrates how to systematically build a graph that achieves optimal qubit overhead while preserving all necessary properties for logical operator deformation.

*Remark* (Remark 12: Worst-Case Graph Construction). We outline a construction that produces a graph  $G$  with qubit overhead  $O(W \log^2 W)$  satisfying all desiderata from Remark 11, for a logical operator  $L$  of weight  $W$ .

Let  $L$  denote the support of  $L$  (so  $|L| = W$ ), and let  $V = L$  be the vertex set.

**Step 1: Z-type support matching.** For each check  $s_j$  with  $S_Z(s_j) \cap L \neq \emptyset$ , the intersection has even cardinality (since  $s_j$  commutes with  $L = \prod_v X_v$ ). Pick a perfect matching and add edges, achieving Desideratum 1 with  $D = 1$ .

**Step 2: Achieve expansion.** Add edges until  $h(G) \geq 1$  (Cheeger constant), satisfying Desideratum 2. The resulting graph  $G_0$  is constant-degree.

**Step 3: Cycle sparsification.** Apply Definition 6 to  $G_0$  with  $R = O(\log^2 W)$  layers (Lemma 2). All generating cycles have weight  $\leq 4$ , satisfying Desideratum 3.

**Result:**

- Total qubits:  $O(W) \cdot O(\log^2 W) = O(W \log^2 W)$ .
- All checks have bounded weight.
- Distance is preserved:  $d^* \geq d$ .

The construction rests on several key mathematical properties. The first step exploits the commutation relation between stabilizer checks and logical operators, which forces the  $Z$ -support intersections to have even cardinality. This enables perfect matching and ensures short deformation paths.

The theoretical foundation begins with understanding how logical operators interact with stabilizer checks through the symplectic inner product. For a logical operator  $L = \prod_{v \in V} X_v$ , any stabilizer check  $s$  must commute with  $L$ , which constrains the structure of the  $Z$ -support.

**Theorem** (Symplectic Inner Product with  $\prod X$ ). *Let  $V$  be a finite type and let  $P$  be a Pauli operator on  $V$ . Then*

$$\langle P, \text{prodX}(V) \rangle_{\text{symp}} = \sum_{v \in V} P \cdot \text{zVec}(v).$$

*Proof.* Expanding the definition of the symplectic inner product and  $\text{prodX}$ , the  $X$ -vector of  $\text{prodX}(V)$  is the all-ones function and the  $Z$ -vector is identically zero. Thus the sum reduces to  $\sum_v P \cdot \text{zVec}(v) \cdot 1 + P \cdot \text{xVec}(v) \cdot 0 = \sum_v P \cdot \text{zVec}(v)$ . The result follows by commutativity of addition.  $\square$

This symplectic characterization immediately yields the equivalence between Pauli commutation and the logical commutation condition.

**Theorem** (Commutation with  $\prod X \Leftrightarrow$  Logical Commutation). *Let  $P$  be a Pauli operator on a finite type  $V$ . Then*

$$\text{PauliCommute}(P, \text{prodX}(V)) \iff \text{CommutesWithLogical}(P).$$

*Proof.* Unfolding the definitions of  $\text{PauliCommute}$  and  $\text{CommutesWithLogical}$ , we use the identity  $\langle P, \text{prodX}(V) \rangle_{\text{symp}} = \sum_v P \cdot \text{zVec}(v)$  from the previous theorem. The equality  $\sum_v P \cdot \text{zVec}(v) = \sum_v \text{zSupportOnVertices}(P)(v)$  completes the equivalence.  $\square$

The crucial constraint on  $Z$ -support cardinality follows from this commutation requirement.

**Theorem** (Commuting Check Has Even  $Z$ -Support). *Let  $P$  be a Pauli operator on a finite type  $V$ . If  $\text{PauliCommute}(P, \text{prodX}(V))$ , then*

$$|\{v \in V \mid P \cdot \text{zVec}(v) \neq 0\}| \text{ is even.}$$

*Proof.* By the equivalence  $\text{PauliCommute}(P, \text{prodX}(V)) \Leftrightarrow \text{CommutesWithLogical}(P)$ , the result follows from the general theorem that logical commutation implies even  $Z$ -support cardinality.  $\square$

With the even cardinality established, Step 1 of the construction creates a graph where each check's  $Z$ -support forms a clique, ensuring unit-distance deformation paths.

**Theorem** (Step 1 Achieves Desideratum 1). *Let  $G$  be a simple graph on a finite type  $V$ , and let  $\{\text{checks}(j)\}_{j \in J}$  be a family of Pauli operators. Suppose that for every check  $j$  and every pair of distinct vertices  $u, v$  in  $\text{supportZ}(\text{checks}(j))$ ,  $u$  and  $v$  are adjacent in  $G$ . Then  $\text{ShortPathsForDeformation}(G, \text{checks}, 1)$  holds.*

*Proof.* For any check  $j$  and vertices  $u, v \in \text{supportZ}(\text{checks}(j))$ , if  $u = v$  then  $\text{dist}(u, v) = 0 \leq 1$ . If  $u \neq v$ , the hypothesis ensures  $G \cdot \text{Adj}(u, v)$ , so  $\text{dist}(u, v) = 1$ .  $\square$

Step 2 adds edges to achieve expansion, which is essential for preserving the code distance.

**Theorem** (Distance Preserved by Expansion). *Let  $G$  be a simple graph on  $V$  with sufficient expansion. For every nonempty subset  $S \subseteq V$  with  $2|S| \leq |V|$ :*

$$|S| \leq |\partial_G S|,$$

where  $\partial_G S$  is the edge boundary of  $S$  in  $G$ .

*Proof.* This follows directly from the definition of sufficient expansion and the expansion-boundary relationship.  $\square$

Step 3 applies cycle sparsification to ensure that all generating cycles have bounded weight, completing the construction.

**Theorem** (Construction Satisfies All Desiderata). *Let  $G$  be a simple graph on  $V$  with the properties established in Steps 1-3. Then the graph simultaneously satisfies:*

1. Short deformation paths with  $D = 1$
2. Sufficient expansion for distance preservation
3. Low-weight cycle basis with cycle weight  $\leq 4$
4. Constant degree bounded by the expansion parameter

*Proof.* Each desideratum follows from the corresponding construction step: Step 1 ensures short paths, Step 2 provides expansion, Step 3 gives bounded cycle weights, and the degree bound follows from the expansion construction.  $\square$

The total qubit overhead achieves the stated  $O(W \log^2 W)$  bound through careful analysis of the sparsification process.

**Theorem** (Total Overhead is  $O(W \log^2 W)$ ). *For the constructed graph with  $W$  logical qubits and  $R = O(\log^2 W)$  sparsification layers:*

$$\text{Total qubits} = (R + 1) \cdot W = O(W \log^2 W).$$

*Proof.* The sparsified vertex count is  $(R + 1) \cdot W$ , where  $R \leq K \cdot (\log_2 W)^2$  for some constant  $K$ . Thus the total count is bounded by  $(K \cdot (\log_2 W)^2 + 1) \cdot W = O(W \log^2 W)$ .  $\square$

This construction demonstrates that the worst-case qubit overhead for logical operator deformation can be achieved constructively, providing both theoretical bounds and a practical algorithmic approach. The key insight is that the commutation constraints naturally lead to even  $Z$ -support intersections, enabling efficient graph constructions that preserve all necessary quantum error correction properties.

### 1.23 Lemma 3: SpaceDistance

Graph expansion plays a crucial role in quantum error correction, particularly in the context of code construction through gauging procedures. When we gauge a quantum code by introducing auxiliary edge qubits on a graph, the resulting deformed code's properties depend intimately on the expansion characteristics of the underlying graph. A key question is whether the distance of the original code is preserved under this gauging transformation.

The distance preservation problem reduces to analyzing how logical operators of the deformed code relate to logical operators of the original code through a "cleaning" procedure. This cleaning process removes unwanted  $X$ -support on edge qubits by multiplying with carefully chosen stabilizer elements, allowing us to extract a logical operator on the vertex qubits that corresponds to the original code space.

**Lemma** (Lemma 3: Space Distance Preservation). *Let  $G = (V, E)$  be a connected graph with  $|V| \geq 2$ , and let  $h(G)$  denote its Cheeger constant (expansion parameter). Consider a quantum stabilizer code with distance  $d$  that has been deformed through gauging on graph  $G$ , producing a deformed code with distance  $d^*$ . Assume:*

1. *Both the first cohomology sequence  $\ker(\delta_2) \subseteq \text{im}(\delta)$  and second boundary exactness  $\ker(\partial) \subseteq \text{im}(\partial_2)$  hold,*
2. *The logical operator  $\text{logicalOpV} = \prod_{v \in V} X_v$  is a logical operator of the original code,*
3. *The deformed code has at least one logical operator.*

*Then the distances satisfy:*

$$d^* \geq \min(h(G), 1) \cdot d.$$

*In particular, if  $h(G) \geq 1$  (sufficient expansion), then  $d^* \geq d$ .*

*Proof.* The proof proceeds through several key steps, establishing the relationship between logical operators of the deformed and original codes via the cleaning procedure.

**Step 1: Edge X-support is a cocycle.** Let  $L'$  be any logical operator of the deformed code. Since  $L'$  commutes with all flux checks  $B_p$ , and the symplectic inner product with flux checks equals the second coboundary applied to the edge X-support, we have  $\delta_2(\text{edgeXSupport}(L')) = 0$ . Thus the edge X-support lies in  $\ker(\delta_2)$ .

**Step 2: Existence of cleaning vector.** By the first cohomology exactness assumption, there exists some  $c_0 : V \rightarrow \mathbb{Z}/2\mathbb{Z}$  with  $\delta(c_0) = \text{edgeXSupport}(L')$ . The cleaned operator  $\text{cleanedOp}(L', c_0) = L' \cdot \text{gaussSubsetProduct}(G, c_0)$  has no X-support on edges and its restriction to vertex qubits gives a logical operator of the original code.

If  $2|\text{supp}(c_0)| > |V|$ , we instead use  $c_1 = c_0 + \mathbf{1}$ , which satisfies  $\delta(c_1) = \delta(c_0)$  (since  $\delta(\mathbf{1}) = 0$ ) but has smaller support due to the partition  $|\text{supp}(c_0)| + |\text{supp}(c_1)| = |V|$ .

**Step 3: Weight decomposition and bounds.** For the chosen cleaning vector  $c$  with  $2|\text{supp}(c)| \leq |V|$ , the weight of  $L'$  decomposes as:

$$\text{wt}(L') = \text{wt}(\text{restrictToV}(L')) + |\text{edge support}|.$$

Since the cleaned restriction is a logical of the original code, its weight is at least  $d$ . The cleaning process ensures:

$$\text{wt}(\text{restrictToV}(L')) + |\text{supp}(c)| \geq d.$$

**Step 4: Expansion constraint.** The Cheeger constant provides the key constraint linking the support size to the edge contribution:

$$h(G) \cdot |\text{supp}(c)| \leq |\partial_E(\text{supp}(c))| = |\text{coboundary support}| \leq |\text{edge support}|.$$

**Step 5: Arithmetic conclusion.** Combining the weight bound, expansion constraint, and weight decomposition, we apply the core arithmetic inequality: if  $w + s \geq d + e$ ,  $h \cdot s \leq e$ , and  $e \leq w$ , then  $w \geq \min(h, 1) \cdot d$ .

Setting  $w = \text{wt}(L')$ ,  $s = |\text{supp}(c)|$ , and  $e = |\text{edge support}|$ , all conditions are satisfied, yielding  $\text{wt}(L') \geq \min(h(G), 1) \cdot d$ .

Since this bound holds for every logical operator  $L'$  of the deformed code, taking the minimum over all such operators gives  $d^* \geq \min(h(G), 1) \cdot d$ .

When  $h(G) \geq 1$ , we have  $\min(h(G), 1) = 1$ , so  $d^* \geq d$ , establishing that sufficient expansion preserves the code distance.  $\square$

This result demonstrates that graph expansion directly controls distance preservation in gauged quantum codes. The Cheeger constant  $h(G)$  acts as a bottleneck: graphs with  $h(G) \geq 1$  provide sufficient expansion to maintain the original code distance, while graphs with smaller Cheeger constants may reduce the distance by a factor of  $h(G)$ . This connection between discrete geometry and quantum error correction provides a principled approach to selecting graphs for code construction procedures.

### 1.24 Remark 13: OptimalCheegerConstant

In the study of deformed code constructions, a natural question arises: what is the optimal value of the Cheeger constant  $h(G)$  for preserving code distance? The deformed code construction modifies an original stabilizer code by introducing auxiliary edge qubits on a graph  $G$ , and the expansion properties of this graph—measured by the Cheeger constant—directly affect the distance of the resulting code.

This remark establishes that  $h(G) = 1$  is indeed optimal for distance preservation. We show that (1)  $h(G) \geq 1$  is sufficient to guarantee  $d' \geq d$ , (2)  $h(G) > 1$  provides no additional benefit since we always have  $d' \leq d$ , yielding  $d' = d$ , and (3)  $h(G) < 1$  causes distance loss with  $d' \geq h(G) \cdot d < d$ . The analysis relies on a key technical tool: the ability to "lift" Pauli operators from the original vertex space to the extended qubit space while preserving their essential properties.

*Remark* (Remark 13: Optimal Cheeger Constant). The Cheeger constant  $h(G) = 1$  is optimal for distance preservation in the deformed code construction. For  $h(G) \geq 1$ , we achieve perfect distance preservation  $d' = d$ , while  $h(G) < 1$  causes distance loss  $d' \geq h(G) \cdot d < d$ .

The proof strategy involves constructing a lifting map that embeds original Pauli operators into the extended qubit space, then analyzing how this map interacts with the deformed code structure. We begin by establishing the basic properties of this lifting construction.

**Definition** (Definition: Lift to Extended System). Given a Pauli operator  $P$  on the vertex space  $V$ , define its **lift** to the extended qubit space  $V \oplus E(G)$  by

$$\text{liftToExtended}(P) := \text{deformedOpExt}(P, 0),$$

where  $P$  acts on vertex qubits as before and acts as the identity on all edge qubits.

The lifting operation preserves the algebraic structure while extending operators to the larger space. Most importantly, it preserves operator weight, which is crucial for distance calculations.

**Lemma** (Lemma: Weight Preservation). *For any Pauli operator  $P$  on  $V$ ,*

$$\text{weight}(\text{liftToExtended}(P)) = \text{weight}(P).$$

*Proof.* We use the vertex-edge weight decomposition formula. The edge contribution is zero: for every edge qubit  $e \in E(G)$ , the lifted operator satisfies both  $\text{xVec}(\text{liftToExtended}(P))(\text{inr}(e)) = 0$  and  $\text{zVec}(\text{liftToExtended}(P))(\text{inr}(e)) = 0$  by construction. Therefore, the set of edges with non-trivial support is empty and contributes 0 to the weight. The vertex contribution equals  $\text{weight}(P)$  by definition, yielding the result.  $\square$

**Lemma** (Lemma: Restriction Recovery). *For any Pauli operator  $P$  on  $V$ ,*

$$\text{restrictToV}(\text{liftToExtended}(P)) = P.$$

*Proof.* By extensionality on each vertex  $v$ , both the  $x$ -vector and  $z$ -vector components agree by direct computation using the definitions.  $\square$

The lift operation respects the group structure of Pauli operators and provides an embedding of the original operator algebra into the extended space.

**Lemma** (Lemma: Multiplicativity and Identity). *The lift operation satisfies:*

1.  $\text{liftToExtended}(P \cdot Q) = \text{liftToExtended}(P) \cdot \text{liftToExtended}(Q)$
2.  $\text{liftToExtended}(\mathbf{1}) = \mathbf{1}$
3. *If  $P \neq \mathbf{1}$ , then  $\text{liftToExtended}(P) \neq \mathbf{1}$  (injectivity)*

*Proof.* Properties (1) and (2) follow by extensionality and direct computation. For (3), suppose  $\text{liftToExtended}(P) = \mathbf{1}$ . Applying restriction to both sides and using the recovery lemma gives  $P = \text{restrictToV}(\mathbf{1}) = \mathbf{1}$ , contradicting the hypothesis.  $\square$

A critical property is that lifted operators interact well with the deformed code structure. Pure- $X$  operators, which have no  $Z$  components, exhibit particularly nice behavior.

**Lemma** (Lemma: Commutation with Deformed Checks). *Let  $P$  be a Pauli operator on  $V$ .*

1.  $\text{liftToExtended}(P)$  commutes with all flux checks.
2. *If  $P$  commutes with an original check, then  $\text{liftToExtended}(P)$  commutes with the corresponding deformed original check.*
3.  $\text{liftToExtended}(P)$  commutes with the Gauss law check  $A_v$  if and only if  $P$  has no  $Z$  support at vertex  $v$ .

*Proof.* For (1), we compute the symplectic inner product by decomposing over vertices and edges. Since flux checks are pure  $Z$  on edges with trivial  $x$ -components on vertices, and lifted operators have zero support on edges, all terms vanish.

For (2), we use that symplectic inner products with deformed checks reduce to the restriction on  $V$  when there is no edge  $X$  support. The result follows from the hypothesis and the recovery property.

For (3), direct computation shows  $\langle \text{liftToExtended}(P), A_v \rangle = P \cdot z\text{Vec}(v)$ , which equals zero if and only if  $P$  has no  $Z$  support at  $v$ .  $\square$

**Theorem** (Theorem: Pure- $X$  Centralization). *Let  $P$  be a pure- $X$  Pauli operator on  $V$  that commutes with all original checks. Then  $\text{liftToExtended}(P)$  lies in the centralizer of the deformed stabilizer code.*

*Proof.* We verify commutation with each type of deformed code check:

- **Gauss law checks:** Since  $P$  is pure- $X$ , it has  $z\text{Vec} = 0$ , so commutation follows from the previous lemma.
- **Flux checks:** Commutation follows directly from the commutation lemma.
- **Deformed original checks:** Since  $P$  commutes with the original checks, commutation with their deformed versions follows from the commutation lemma.

□

Now we establish the three main points about optimal expansion. Point 1 shows that sufficient expansion ( $h(G) \geq 1$ ) guarantees distance preservation.

**Theorem** (Theorem: Point 1 - Sufficient Expansion). *Let  $C$  be a stabilizer code with distance  $d$ , and let  $G$  be a connected graph on  $V$  with  $|V| \geq 2$  satisfying  $h(G) \geq 1$ . Under appropriate exactness conditions, the deformed code distance satisfies  $d^* \geq d$ .*

*Proof.* This follows directly from the general sufficient expansion theorem established earlier in the formalization. □

Point 2 demonstrates that we cannot improve beyond the original distance by showing  $d^* \leq d$  via the lifting construction.

**Theorem** (Theorem: Point 2 - Upper Bound via Lifting). *If there exists a pure- $X$  logical operator  $P$  of the original code with  $\text{weight}(P) = d$  whose lift is not in the deformed stabilizer group, then  $d^* \leq d$ .*

*Proof.* First, we establish that  $\text{liftToExtended}(P)$  is a logical operator of the deformed code. Since  $P$  is pure- $X$  and commutes with all original checks, the centralization theorem shows that the lift lies in the deformed centralizer. By hypothesis, it is not in the deformed stabilizer group, and by injectivity of lifting, it is non-identity. Therefore, it satisfies all conditions for being a logical operator.

The distance bound follows immediately:

$$d^* \leq \text{weight}(\text{liftToExtended}(P)) = \text{weight}(P) = d,$$

where we use the definition of distance as an infimum, weight preservation of lifting, and the hypothesis. □

Combining Points 1 and 2 yields the optimal distance preservation result.

**Theorem** (Theorem: Perfect Distance Preservation). *Under the hypotheses of both Point 1 ( $h(G) \geq 1$ ) and Point 2 (existence of a suitable pure- $X$  minimum-weight logical), the deformed code distance equals the original distance:  $d^* = d$ .*

*Proof.* Apply antisymmetry:  $d' \leq d$  from Point 2 and  $d \leq d'$  from Point 1. □

Point 3 shows that insufficient expansion causes distance loss.

**Theorem** (Theorem: Point 3 - Distance Loss). *If  $h(G) < 1$ , then  $d^* \geq h(G) \cdot d$ . Since  $h(G) < 1$  and  $d > 0$ , this bound  $h(G) \cdot d$  is strictly less than  $d$ , representing genuine distance loss.*

*Proof.* The bound  $d' \geq h(G) \cdot d$  follows from the general theorem that  $d' \geq \min(h(G), 1) \cdot d$  and the fact that  $\min(h(G), 1) = h(G)$  when  $h(G) < 1$ . The strict inequality  $h(G) \cdot d < d$  follows from  $h(G) < 1$  and  $d > 0$ . □

The optimality of  $h(G) = 1$  is now evident from the trichotomy of behaviors.

**Theorem** (Theorem: Distance Trichotomy). *Let  $d > 0$  and  $h(G) > 0$ . The distance behavior exhibits the following dichotomy:*

1. *If  $h(G) \geq 1$ , then the lower bound  $\min(h(G), 1) \cdot d$  equals  $d$  (perfect preservation).*

2. If  $h(G) < 1$ , then the lower bound  $\min(h(G), 1) \cdot d$  is strictly less than  $d$  (distance loss).

*Proof.* For case (1),  $h(G) \geq 1$  implies  $\min(h(G), 1) = 1$ , so the bound becomes  $1 \cdot d = d$ . For case (2),  $h(G) < 1$  implies  $\min(h(G), 1) = h(G)$ , and the strict inequality  $h(G) \cdot d < d$  follows from the positivity conditions.  $\square$

This trichotomy reveals that  $h(G) = 1$  represents a sharp threshold: any  $h(G) \geq 1$  achieves optimal distance preservation, while  $h(G) < 1$  necessarily causes distance loss. The Cheeger constant  $h(G) = 1$  is therefore optimal in the sense that it is both necessary and sufficient for maintaining the original code distance in the deformed construction.

### 1.25 Remark 14: ParallelGaugingMeasurement

The measurement of multiple logical operators in quantum error correction can often be performed in parallel, provided certain compatibility conditions are satisfied. This parallel measurement capability is crucial for achieving the fault-tolerance thresholds required in large-scale quantum computation, where hundreds or thousands of logical operators may need to be measured simultaneously. The key insight is that logical operators with compatible support structures can share measurement resources without compromising the error correction properties of the underlying code.

*Remark* (Remark 14: Parallel Gauging Measurement). The gauging measurement protocol can be applied simultaneously to multiple logical operators  $(L_1, \dots, L_m)$ , subject to four fundamental compatibility conditions:

(1) **Non-overlapping support:** Logical operators with disjoint support sets can be gauged in parallel using independent gauge graphs  $G_1, \dots, G_m$ .

(2) **Same-type overlapping support:** Operators sharing support qubits can be gauged together if they act with the same Pauli type (all  $X$  or all  $Z$ ) on every shared qubit.

(3) **LDPC preservation:** At most a constant number  $c$  of logical operators should have overlapping support on any single qubit to maintain the low-density parity-check property.

(4) **Time-space tradeoff:** The protocol can measure  $2m - 1$  redundant copies across  $\lceil d/m \rceil$  measurement rounds, using majority vote for error resilience, thereby trading increased spatial overhead for reduced temporal complexity.

This remark establishes the theoretical foundation for scalable quantum error correction protocols, where the ability to perform parallel measurements directly impacts the overall computational efficiency. The mathematical framework developed here provides precise conditions under which such parallelization preserves both the logical information and the error correction capabilities of the quantum code.

### 1.26 Definition 7: SpaceAndTimeFaults

In fault-tolerant quantum error correction, errors can occur in two fundamental ways: as physical errors on qubits during quantum operations (space-faults) and as errors in measurement outcomes or state preparations (time-faults). Understanding and formalizing these error models is crucial for analyzing the performance of quantum error correction codes and developing effective decoding strategies.

The distinction between space and time errors reflects the dual nature of quantum information processing, where we must account for both unitary evolution errors and measurement/preparation

errors. This comprehensive error model allows us to study realistic fault scenarios where both types of errors can occur simultaneously.

**Definition** (Definition 7: Space and Time Faults). A **space-fault** on qubit set  $Q$  at times  $T$  is a structure consisting of:

- a qubit  $q \in Q$  where the error occurs,
- a time  $t \in T$  at which the error occurs,
- an  $X$ -component  $x \in \mathbb{Z}/2\mathbb{Z}$  (equal to 1 if the error has an  $X$  or  $Y$  component, 0 otherwise),
- a  $Z$ -component  $z \in \mathbb{Z}/2\mathbb{Z}$  (equal to 1 if the error has a  $Z$  or  $Y$  component, 0 otherwise),
- a nontriviality condition:  $x \neq 0$  or  $z \neq 0$ .

A **time-fault** on measurement set  $M$  is a structure consisting of:

- a measurement  $m \in M$  whose outcome is reported incorrectly.

A **spacetime fault** on qubit set  $Q$ , time set  $T$ , and measurement set  $M$  is a structure consisting of:

- a finite set  $\text{spaceFaults} \subseteq \text{Finset}(\text{SpaceFault}(Q, T))$  of space-faults,
- a finite set  $\text{timeFaults} \subseteq \text{Finset}(\text{TimeFault}(M))$  of time-faults.

The **weight** of a spacetime fault  $F$  is:

$$\text{weight}(F) = |\text{spaceFaults}(F)| + |\text{timeFaults}(F)|.$$

The **composition** of two spacetime faults  $F_1$  and  $F_2$  is defined via symmetric difference:

$$\text{compose}(F_1, F_2) = (\text{spaceFaults}(F_1) \Delta \text{spaceFaults}(F_2), \text{timeFaults}(F_1) \Delta \text{timeFaults}(F_2)).$$

This formalization captures several key properties. Space-faults encode single-qubit Pauli errors ( $I$ ,  $X$ ,  $Y$ ,  $Z$ ) occurring at specific locations and times, with the nontriviality condition ensuring we only consider actual errors (not the identity). Time-faults model measurement outcome flips and can also represent state initialization errors. The composition operation reflects the fact that applying the same error twice cancels it, making the spacetime faults form an abelian group under composition with the empty fault as identity.

The weight function provides a natural measure of error complexity, while the symmetric difference composition ensures that spacetime faults form a mathematical structure suitable for error correction analysis, where error combinations and cancellations are handled algebraically.

## 1.27 Definition 8: Detectors

In quantum error correction, the ability to detect errors in quantum computations is fundamental to maintaining quantum information integrity. While quantum measurements destroy superposition states, classical processing of measurement outcomes can reveal the presence of errors through carefully designed detection schemes. This leads us to the concept of detectors, which are algebraic structures that encode relationships between measurement outcomes and enable systematic error detection.

The key insight is that in a fault-free quantum computation, certain linear combinations of measurement outcomes must always yield predictable parities. When these expected relationships are violated, it signals the presence of errors that have corrupted either the quantum state or the measurement process itself.

**Definition** (Definition 8: Detectors). A **detector** is a structure  $(D, f, c)$  where:

- $D \subseteq M$  is a finite set of measurement labels,
- $f : M \rightarrow \mathbb{Z}/2\mathbb{Z}$  assigns to each measurement its ideal outcome ( $0 \leftrightarrow +1, 1 \leftrightarrow -1$ ),
- $c$  is a proof of the *detector constraint*:  $\sum_{m \in D} f(m) = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ .

In the  $\{+1, -1\}$  encoding, the constraint states that the product of ideal outcomes over the detector's measurements equals  $+1$ .

The detector constraint ensures that in the absence of faults, the parity of actual measurement outcomes will always be even (corresponding to a product of  $+1$ ). This provides a baseline against which deviations can be detected, forming the mathematical foundation for error syndrome extraction in quantum error correction protocols.

**Definition** (Observed Parity). Given a detector  $D$  and a set of time-faults  $F$ , the **observed parity** is defined as

$$\text{observedParity}(D, F) = \sum_{m \in D.\text{measurements}} \text{observedOutcome}(D.\text{idealOutcome}, F, m) \in \mathbb{Z}/2\mathbb{Z}.$$

In  $\{+1, -1\}$  encoding,  $0$  means the product of observed outcomes is  $+1$ , and  $1$  means it is  $-1$ .

**Definition** (Detector Violation). A detector  $D$  is **violated** by a set of time-faults  $F$  if the observed parity equals  $1$ :

$$\text{isViolated}(D, F) \iff \text{observedParity}(D, F) = 1.$$

**Definition** (Flip Parity). The **flip parity** of a detector  $D$  with respect to faults  $F$  is the sum in  $\mathbb{Z}/2\mathbb{Z}$  of the indicator of faulted measurements in  $D$ :

$$\text{flipParity}(D, F) = \sum_{m \in D.\text{measurements}} \begin{cases} 1 & \text{if } \langle m \rangle \in F \\ 0 & \text{otherwise} \end{cases}.$$

**Theorem** (No-Fault Observed Parity). *In the absence of faults, the observed parity of any detector equals  $0$ :*

$$\text{observedParity}(D, \emptyset) = 0.$$

*Proof.* When no faults are present, for every  $m \in D.\text{measurements}$  we have  $\langle m \rangle \notin F$ , so the observed outcome reduces to the ideal outcome  $D.\text{idealOutcome}(m)$ . Thus the observed parity becomes  $\sum_{m \in D.\text{measurements}} D.\text{idealOutcome}(m)$ , which equals  $0$  by the detector constraint.  $\square$

**Theorem** (No Violation Without Faults). *In the absence of faults, no detector is violated:*

$$\neg \text{isViolated}(D, \emptyset).$$

*Proof.* By definition, violation requires  $\text{observedParity}(D, \emptyset) = 1$ . However, by the no-fault observed parity theorem, we have  $\text{observedParity}(D, \emptyset) = 0$ , and  $0 \neq 1$ .  $\square$

**Theorem** (Observed Parity Equals Flip Parity). *The observed parity equals the flip parity:*

$$\text{observedParity}(D, F) = \text{flipParity}(D, F).$$

*That is, the observed parity reduces to counting (mod 2) how many of the detector's measurements are faulted, because the ideal outcomes cancel by the detector constraint.*

*Proof.* The observed parity expands as  $\sum_{m \in D.\text{measurements}} (D.\text{idealOutcome}(m) + \mathbf{1}_{\langle m \rangle \in F})$ . By distributivity of summation, this equals:

$$\sum_{m \in D.\text{measurements}} D.\text{idealOutcome}(m) + \sum_{m \in D.\text{measurements}} \mathbf{1}_{\langle m \rangle \in F}.$$

By the detector constraint, the first sum equals 0, leaving only the flip parity term.  $\square$

**Theorem** (Violation Iff Flip Parity One). *A detector is violated if and only if its flip parity equals 1:*

$$\text{isViolated}(D, F) \iff \text{flipParity}(D, F) = 1.$$

*Proof.* This follows immediately from the definitions of violation and the equality  $\text{observedParity}(D, F) = \text{flipParity}(D, F)$ .  $\square$

**Theorem** (Flip Parity Without Faults). *The flip parity with no faults is 0:*

$$\text{flipParity}(D, \emptyset) = 0.$$

*Proof.* Since no measurement label belongs to the empty fault set  $\emptyset$ , every indicator  $\mathbf{1}_{\langle m \rangle \in \emptyset}$  equals 0, making the entire sum equal to 0.  $\square$

**Theorem** (Violation Depends on Intersection). *The violation of a detector depends only on which of the detector's measurements appear in the fault set. If two fault sets  $F_1, F_2$  satisfy  $\langle m \rangle \in F_1 \iff \langle m \rangle \in F_2$  for all  $m \in D.\text{measurements}$ , then*

$$\text{isViolated}(D, F_1) \iff \text{isViolated}(D, F_2).$$

*Proof.* By the characterization of violation in terms of flip parity, it suffices to show  $\text{flipParity}(D, F_1) = \text{flipParity}(D, F_2)$ . For each  $m \in D.\text{measurements}$ , the indicators  $\mathbf{1}_{\langle m \rangle \in F_1}$  and  $\mathbf{1}_{\langle m \rangle \in F_2}$  are equal by the given hypothesis, so the sums are equal.  $\square$

**Theorem** (Violation Invariant Under Disjoint Faults). *Violation is invariant if we add faults outside the detector. If no measurement of  $D$  appears in  $\text{extra}$ , then*

$$\text{isViolated}(D, F \cup \text{extra}) \iff \text{isViolated}(D, F).$$

*Proof.* We apply the intersection-dependence theorem. For each  $m \in D.\text{measurements}$ , we show  $\langle m \rangle \in F \cup \text{extra} \iff \langle m \rangle \in F$ . If  $\langle m \rangle \in F \cup \text{extra}$ , then either  $\langle m \rangle \in F$  or  $\langle m \rangle \in \text{extra}$ . The latter contradicts the disjointness hypothesis, so  $\langle m \rangle \in F$ . Conversely, if  $\langle m \rangle \in F$ , then  $\langle m \rangle \in F \cup \text{extra}$  by set inclusion.  $\square$

**Definition** (Repeated Measurement Detector). A **repeated measurement detector** is formed from two consecutive measurements  $m_1 \neq m_2$  of the same stabilizer check with common ideal outcome  $o \in \mathbb{Z}/2\mathbb{Z}$ . The detector has:

- measurements =  $\{m_1, m_2\}$ ,
- ideal outcome  $f(m) = o$  if  $m = m_1$  or  $m = m_2$ , and 0 otherwise.

The detector constraint holds because  $o + o = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ .

**Definition** (Initialization-Measurement Detector). An **initialization-measurement detector** is formed from an initialization event  $m_{\text{init}}$  and a later measurement  $m_{\text{meas}}$  with  $m_{\text{init}} \neq m_{\text{meas}}$ , both having the same ideal outcome  $o$ . This forms a valid detector with measurements  $\{m_{\text{init}}, m_{\text{meas}}\}$  and ideal outcome  $o$  on both, satisfying the constraint  $o + o = 0$  in  $\mathbb{Z}/2\mathbb{Z}$ .

**Theorem** (Repeated Measurement Violation Characterization). *A repeated measurement detector on  $m_1, m_2$  with  $m_1 \neq m_2$  is violated by faults  $F$  if and only if exactly one of the two measurements is faulted:*

$$\text{isViolated}(\text{repeatedMeasurementDetector}(m_1, m_2), F) \iff (\langle m_1 \rangle \in F) \oplus (\langle m_2 \rangle \in F).$$

*Proof.* Using the flip parity characterization, the detector is violated iff  $\mathbf{1}_{\langle m_1 \rangle \in F} + \mathbf{1}_{\langle m_2 \rangle \in F} = 1$  in  $\mathbb{Z}/2\mathbb{Z}$ . This sum equals 1 if and only if exactly one of the indicators is 1, which occurs precisely when exactly one of  $\langle m_1 \rangle \in F$  or  $\langle m_2 \rangle \in F$  holds.  $\square$

**Definition** (Syndrome). The **syndrome** of a set of time-faults  $F$  with respect to a family of detectors  $(D_i)_{i \in I}$  is the set of detector indices whose detectors are violated:

$$\text{syndrome}(\{D_i\}, F) = \{i \in I \mid \text{observedParity}(D_i, F) = 1\}.$$

**Theorem** (Membership in Syndrome). *A detector index  $i$  is in the syndrome if and only if the detector  $D_i$  is violated:*

$$i \in \text{syndrome}(\{D_i\}, F) \iff \text{isViolated}(D_i, F).$$

*Proof.* This follows directly from the definitions of syndrome and violation.  $\square$

**Theorem** (Empty Syndrome Without Faults). *The syndrome is empty when there are no faults:*

$$\text{syndrome}(\{D_i\}, \emptyset) = \emptyset.$$

*Proof.* For each detector index  $i$ , we have  $\text{observedParity}(D_i, \emptyset) = 0$  by the no-fault observed parity theorem, so  $i$  is not in the syndrome since  $0 \neq 1$ .  $\square$

**Definition** (Single-Measurement Detector). A **single-measurement detector** for measurement  $m$  has measurements =  $\{m\}$  and ideal outcome constantly 0. The constraint  $\sum_{m' \in \{m\}} 0 = 0$  holds trivially. This detector fires when measurement  $m$  is faulted.

**Theorem** (Single-Measurement Violation Characterization). *A single-measurement detector on  $m$  is violated if and only if the measurement  $m$  is faulted:*

$$\text{isViolated}(\text{singleMeasurementDetector}(m), F) \iff \langle m \rangle \in F.$$

*Proof.* By the flip parity characterization, violation occurs iff  $\text{flipParity}(\text{singleMeasurementDetector}(m), F) = 1$ . The flip parity over the singleton  $\{m\}$  equals  $\mathbf{1}_{\langle m \rangle \in F}$ , which equals 1 if and only if  $\langle m \rangle \in F$ .  $\square$

**Definition** (Detector Weight). The **weight** of a detector  $D$  is the number of measurements it contains:

$$\text{detectorWeight}(D) = |D.\text{measurements}|.$$

**Theorem** (Weight Properties). *The weights of standard detector types are:*

$$\text{detectorWeight}(\text{repeatedMeasurementDetector}(m_1, m_2)) = 2 \quad (7)$$

$$\text{detectorWeight}(\text{initAndMeasureDetector}(m_{\text{init}}, m_{\text{meas}})) = 2 \quad (8)$$

$$\text{detectorWeight}(\text{singleMeasurementDetector}(m)) = 1 \quad (9)$$

$$\text{detectorWeight}(\text{emptyDetector}) = 0 \quad (10)$$

*Proof.* Each follows by counting the elements in the respective measurement sets, using the distinctness conditions where applicable.  $\square$

## 1.28 Definition 9: Syndrome

In quantum error correction, the syndrome serves as the crucial diagnostic tool that reveals which detectors have been triggered by a fault. When a spacetime fault occurs, it may cause certain measurements to produce unexpected outcomes, thus violating the stabilizer constraints encoded by the detectors. The syndrome captures exactly this violation pattern, providing the error correction protocol with the information needed to identify and correct the underlying fault.

Since detectors depend only on measurement outcomes and not on the underlying physical qubits themselves, only the time-fault component of a spacetime fault affects the syndrome. This observation leads to a natural representation of syndromes both as finite sets of violated detector indices and as binary vectors over  $\mathbb{Z}_2$ .

**Definition** (Definition 9: Syndrome). Let  $I$  be a finite type indexing a collection of detectors  $\{D_i\}_{i \in I}$ , and let  $F$  be a spacetime fault. The **syndrome fault set** of  $F$  with respect to the detectors is the finite set of detector indices that are violated by  $F$ :

$$\text{syndromeFault}(\{D_i\}, F) = \{i \in I \mid D_i \text{ is violated by } F.\text{timeFaults}\}.$$

The **syndrome vector** of a spacetime fault  $F$  with respect to detectors  $\{D_i\}_{i \in I}$  is the binary vector  $s: I \rightarrow \mathbb{Z}_2$  defined by

$$s(i) = \begin{cases} 1 & \text{if } D_i \text{ is violated by } F.\text{timeFaults}, \\ 0 & \text{otherwise.} \end{cases}$$

The syndrome provides a complete characterization of which detectors are violated, with the finite set representation being natural for theoretical analysis and the vector representation being convenient for computational purposes. These two representations are equivalent and capture the same information about the fault pattern.

**Theorem** (Theorem: Syndrome Membership Characterization). *A detector index  $i$  is in the syndrome if and only if detector  $D_i$  is violated by the spacetime fault:*

$$i \in \text{syndromeFault}(\{D_k\}, F) \Leftrightarrow D_i.\text{isViolated}(F.\text{timeFaults}).$$

*Proof.* By the definition of `syndromeFault` as a filter over the universal finite set, the result follows by simplification.  $\square$

**Theorem** (Theorem: Syndrome Vector Characterization). *The syndrome vector satisfies  $s(i) = 1$  if and only if detector  $D_i$  is violated:*

$$\text{syndromeVec}(\{D_k\}, F)(i) = 1 \Leftrightarrow D_i.\text{isViolated}(F.\text{timeFaults}).$$

*Proof.* Unfolding the definition of `syndromeVec`, we split on the conditional. If the detector is violated, then  $s(i) = 1$  by definition, and the forward direction is trivially satisfied. If the detector is not violated, then  $s(i) = 0$ , so  $s(i) = 1$  is contradicted by  $0 \neq 1$ , and the backward direction is contradicted by the assumption that the detector is not violated.  $\square$

**Theorem** (Theorem: Spacetime Syndrome Equals Detector Syndrome). *The spacetime syndrome equals the detector syndrome applied to the time-fault component:*

$$\text{syndromeFault}(\{D_k\}, F) = \text{Detector}.\text{syndrome}(\{D_k\}, F.\text{timeFaults}).$$

*Proof.* By extensionality on the index  $i$ , we simplify using the definitions of `syndromeFault`, `Detector.syndrome`, and `Detector.isViolated`. Both sides reduce to filtering the universal set by the same predicate.  $\square$

**Theorem** (Theorem: Fault-Free Configuration Properties). *The syndrome is empty for the fault-free spacetime configuration:*

$$\text{syndromeFault}(\{D_k\}, \text{empty}) = \emptyset$$

and the syndrome vector is zero:

$$\text{syndromeVec}(\{D_k\}, \text{empty}) = 0.$$

*Proof.* For the first statement, simplifying using the definition of `syndromeFault`, the filter is empty if no detector is violated. For each detector index  $i$ , this follows from the fact that no detector is violated in the absence of faults.

For the second statement, by extensionality on the index  $i$ , we simplify the definition of `syndromeVec` at the empty fault. Since no detector is violated in the absence of faults, the conditional evaluates to 0, which equals the zero function applied at  $i$ .  $\square$

**Theorem** (Theorem: Syndrome Depends Only on Time-Faults). *Space-faults do not affect the syndrome. If  $F_1.\text{timeFaults} = F_2.\text{timeFaults}$ , then*

$$\text{syndromeFault}(\{D_k\}, F_1) = \text{syndromeFault}(\{D_k\}, F_2)$$

and

$$\text{syndromeVec}(\{D_k\}, F_1) = \text{syndromeVec}(\{D_k\}, F_2).$$

*Proof.* For both statements, we proceed by extensionality on the index  $i$ . We simplify using the definitions of the respective syndrome functions, `Detector.isViolated`, and `Detector.observedParity`, together with the hypothesis that the time-fault components are equal. Both sides reduce to the same expression.  $\square$

**Theorem** (Theorem: Syndrome Equivalence Between Representations). *The syndrome finite set is exactly the support of the syndrome vector:*

$$\text{syndromeFault}(\{D_k\}, F) = \{i \in I \mid \text{syndromeVec}(\{D_k\}, F)(i) = 1\}.$$

Consequently, membership relationships hold:

$$i \in \text{syndromeFault}(\{D_k\}, F) \Leftrightarrow \text{syndromeVec}(\{D_k\}, F)(i) = 1$$

and

$$i \notin \text{syndromeFault}(\{D_k\}, F) \Leftrightarrow \text{syndromeVec}(\{D_k\}, F)(i) = 0.$$

*Proof.* By extensionality on the index  $i$ , we simplify using the membership characterization of the syndrome finite set and the characterization of the syndrome vector being 1, which both reduce to the detector being violated. The membership relationships follow directly from rewriting using the respective characterizations.  $\square$

The syndrome construction provides a fundamental bridge between the physical layer of quantum error correction (where faults occur) and the classical processing layer (where error correction algorithms operate on binary syndrome data). The key insight that syndromes depend only on measurement outcomes, not on the underlying quantum state, makes efficient classical post-processing possible while maintaining the full error-correcting capability of the quantum code.

### 1.29 Definition 10: FaultTolerantGaugingProcedure

Fault-tolerant quantum error correction requires systematic procedures for measuring stabilizer generators while maintaining error correction capabilities throughout the process. In quantum error correction with gauge degrees of freedom, we need a structured approach to transition between different measurement phases while preserving the logical information. This leads us to define a comprehensive measurement procedure that coordinates the timing and types of measurements across three distinct phases.

**Definition** (Definition 10: Fault-Tolerant Gauging Procedure). The **fault-tolerant gauging measurement procedure** for measuring a logical operator  $L$  in an  $\llbracket n, k, d \rrbracket$  stabilizer code using a connected graph  $G = (V, E)$  with cycle set  $C$  and check set  $J$  is a structure consisting of:

1.  $d \in \mathbb{N}$ , the code distance (number of rounds per phase), with  $d \geq 1$ .
2. A proof that the original stabilizer code checks pairwise commute: for all  $i, j \in J$ , the Pauli operators satisfy  $\text{PauliCommute}(\text{checks}(i), \text{checks}(j))$ .
3. A gauging input specifying the base vertex and connectivity data.
4. Deformed code data: edge-paths satisfying boundary conditions.
5. A cycle parity condition: for each cycle  $c \in C$  and vertex  $v \in V$ , the number of edges in the cycle incident to  $v$  is even.

The procedure operates in three phases:

- **Phase 1 (Pre-Deformation)**: Times  $t \in [0, d)$ , measuring original stabilizer checks
- **Phase 2 (Deformed Code)**: Times  $t \in [d, 2d)$ , measuring Gauss's law, flux, and deformed original checks
- **Phase 3 (Post-Deformation)**: Times  $t \in [2d, 3d)$ , ungauging edge qubits and resuming original checks

The phase assignment function is given by:

$$\text{phaseOf}(d, t) = \begin{cases} \text{preDeformation} & \text{if } t < d \\ \text{deformedCode} & \text{if } d \leq t < 2d \\ \text{postDeformation} & \text{if } 2d \leq t \end{cases}$$

This structured approach ensures that measurements are coordinated across different phases while maintaining the fault-tolerance properties essential for quantum error correction. The three-phase structure allows for a smooth transition from the original stabilizer code to the deformed gauged code and back, with each phase having duration exactly  $d$  rounds to provide sufficient redundancy for error detection and correction.

### 1.30 Lemma 4: SpacetimeCodeDetectors

The fault-tolerant gauging procedure generates a spacetime code whose local structure must be carefully analyzed to understand error detection capabilities. Each phase of the procedure produces characteristic measurement patterns that, when repeated across multiple rounds, create temporal correlations essential for fault tolerance. Understanding the complete set of these local detectors is crucial for establishing the error-correcting properties of the resulting spacetime code.

In quantum error correction, detectors are fundamental objects that capture the constraint structure of measurement outcomes in the absence of errors. For the fault-tolerant gauging procedure, detectors arise from three sources: repeated measurements of self-inverse operators within each phase, boundary conditions at the gauging transitions, and the constraint relationships imposed by the gauge-fixing protocol. The key mathematical question is whether the naturally occurring detectors form a complete generating set under  $\mathbb{Z}_2$  operations.

**Definition** ( $\mathbb{Z}_2$  Generation). Let  $\alpha$  be a type with decidable equality and let  $\{\text{generators}_i\}_{i \in \iota}$  be a family of finsets over  $\alpha$ . A finset  $S$  is  **$\mathbb{Z}_2$ -generated** by the generators if  $S$  can be obtained from  $\emptyset$  by iterated symmetric differences with generators. Formally, this is defined inductively:

- $\emptyset$  is generated.
- If  $S$  is generated, then  $S \Delta \text{generators}_i$  is generated for any  $i \in \iota$ .

**Lemma** (Lemma 4a: Generator Membership). *Each generator  $\text{generators}_i$  is in the  $\mathbb{Z}_2$  span.*

*Proof.* We have  $\text{generators}_i = \emptyset \Delta \text{generators}_i$ . Since  $\emptyset$  is generated (by the base case), the symmetric difference rule gives that  $\emptyset \Delta \text{generators}_i$  is generated. By the identity  $\emptyset \Delta x = x$ , this equals  $\text{generators}_i$ .  $\square$

**Lemma** (Lemma 4b:  $\mathbb{Z}_2$  Span Closure under Symmetric Difference). *If  $S$  and  $T$  are both  $\mathbb{Z}_2$ -generated by the generators, then  $S \Delta T$  is also generated.*

*Proof.* We proceed by induction on the derivation that  $T$  is generated, generalizing over  $S$ .

**Base case** ( $T = \emptyset$ ): We have  $S \Delta \emptyset = S$ , which is generated by assumption.

**Inductive step** ( $T = U \Delta \text{generators}_i$  where  $U$  is generated): By the induction hypothesis applied to  $S$ , we know  $S \Delta U$  is generated. By associativity of symmetric difference,  $S \Delta (U \Delta \text{generators}_i) = (S \Delta U) \Delta \text{generators}_i$ . Since  $S \Delta U$  is generated, the symmetric difference rule gives the result.  $\square$

The detectors in the fault-tolerant gauging procedure fall into several categories based on their temporal and spatial structure. During the original code phases ( $t < t_i$  and  $t > t_o$ ), detectors arise from repeated measurements of checks  $s_j$  at consecutive rounds. During the deformed code phase ( $t_i < t < t_o$ ), detectors come from repeated measurements of deformed code checks  $A_v$ ,  $B_p$ , and  $\tilde{s}_j$ . At the boundaries  $t = t_i$  and  $t = t_o$ , detectors combine initialization/readout events with the first/last measurements of the respective phases.

**Theorem** (Axiom: Generators Span All Detectors). *For any fault-tolerant gauging procedure with  $d \geq 2$ , any detector  $D$  satisfying  $\neg D.isViolated(\emptyset)$  has its measurement set  $\mathbb{Z}_2$ -generated by the generator measurement sets. That is,  $D$ .measurements can be expressed as a symmetric difference of generator measurement sets.*

*This is stated as an axiom (unproven) in the formalization.*

**Justification:** This axiom captures the structural argument that within each phase, the only sources of deterministic measurement constraints are repeated measurements of self-inverse stabilizers and boundary initialization/readout relations. The physical reasoning is that in the absence of faults, measurement outcomes are determined by eigenvalue relationships of commuting operators, leading to predictable constraint patterns.

**Status:** This axiom represents known structural properties of spacetime codes but relies on physical reasoning about eigenvalue outcomes that was not fully formalized in the original development.

**Lemma** (Lemma 4: SpacetimeCodeDetectors). *Assuming  $d \geq 2$  and that every edge qubit is in at least one cycle, the fault-tolerant gauging procedure admits a complete set of local detector generators. These generators include:*

- **Phase 1 repeated detectors:** *For consecutive rounds measuring the same original check  $s_j$*
- **Phase 2 repeated detectors:** *For consecutive rounds measuring Gauss laws  $A_v$ , fluxes  $B_p$ , or deformed checks  $\tilde{s}_j$*
- **Phase 3 repeated detectors:** *For consecutive rounds measuring the same original check  $s_j$*
- **Initialization boundary detectors:** *Relating edge qubit initialization to first Phase 2 measurements*
- **Ungauging boundary detectors:** *Relating last Phase 2 measurements to edge qubit readouts*

*These generators satisfy:*

1. **Validity:** *Every generator is a valid detector (not violated without faults)*
2. **Coverage:** *Every measurement participates in at least one generator*
3.  **$\mathbb{Z}_2$  closure:** *The span is closed under symmetric difference*
4. **Generation:** *Every valid detector decomposes as a  $\mathbb{Z}_2$  combination of the generators*

*Proof.* **This proof relies on the unproven Axiom: Generators Span All Detectors.**

The four properties are established systematically:

**Validity:** Each generator detector corresponds to a physical constraint relationship that holds in the absence of faults. Repeated measurement detectors capture the fact that consecutive measurements of self-inverse operators yield identical eigenvalues. Boundary detectors encode the

initialization and readout protocols where edge qubits start in known states and are measured in the computational basis.

**Coverage:** We verify by exhaustive case analysis that every measurement in the procedure participates in at least one generator. Phase 1 and Phase 3 measurements are covered by repeated detectors pairing consecutive rounds, with boundary detectors handling the first and last rounds. Phase 2 measurements are similarly covered, with the gauging and ungauging boundary detectors handling the interface rounds. Edge initialization and readout measurements are incorporated into the flux and deformed boundary detectors.

$\mathbb{Z}_2$  **Closure:** This follows from the general closure property of  $\mathbb{Z}_2$  generation under symmetric difference operations, established in the preliminary lemmas.

**Generation: By the unproven Axiom: Generators Span All Detectors,** any valid detector has its measurement set expressible as a symmetric difference of generator measurement sets. This axiom relies on the structural argument that measurement constraints arise solely from eigenvalue relationships of commuting operators and boundary conditions.  $\square$

This result establishes that the local detector structure of the spacetime code generated by fault-tolerant gauging has a finite, explicit generating set. The completeness property is essential for understanding the error syndrome space and developing efficient decoding algorithms. However, the proof fundamentally relies on physical reasoning about quantum measurement outcomes that requires the stated axiom for formal completion.

### 1.31 Definition 11: SpacetimeLogicalFault

In quantum error correction for topological codes, the classical dichotomy between correctable errors and logical operations extends to the spacetime setting, where errors occur across both spatial locations and temporal rounds of a measurement protocol. The fault-tolerant implementation of gauge fixing procedures introduces additional complexity, as errors can affect both the measurement outcomes and the logical information encoded in the post-measurement state.

When implementing fault-tolerant gauge fixing, we must distinguish between errors that can be detected and corrected versus those that cause undetectable logical damage. This leads to a fundamental classification of syndrome-free faults based on whether they preserve or alter the measurement procedure's outcome.

**Definition** (Definition 11: Syndrome-Free Fault). A spacetime fault  $F$  is **syndrome-free** with respect to a collection of detectors  $(\delta_i)_{i \in I}$  if the syndrome is empty:

$$\text{syndromeFault}(\delta, F) = \emptyset.$$

Equivalently, no detector is violated by the time-faults of  $F$ .

**Definition** (Definition 11: Spacetime Logical Fault). A **spacetime logical fault** is a spacetime fault  $F$  satisfying:

1.  $\text{IsSyndromeFree}(\delta, F)$ : No detector is violated (syndrome is empty).
2.  $\text{IsOutcomeChanging}(F)$ : The fault changes the measurement result or applies a non-trivial logical operator to the post-measurement state.

**Definition** (Definition 11: Spacetime Stabilizer). A **spacetime stabilizer** is a spacetime fault  $F$  satisfying:

1.  $\text{IsSyndromeFree}(\delta, F)$ : No detector is violated (syndrome is empty).
2.  $\text{IsOutcomePreserving}(F)$ : The fault preserves both the measurement result and the logical information in the post-measurement state.

The key insight is that every syndrome-free fault falls into exactly one of these two categories, establishing a complete dichotomy for undetectable errors. This classification is fundamental for understanding the logical structure of fault-tolerant protocols and determines which syndrome-free error patterns constitute actual logical errors versus harmless stabilizer elements.

**Theorem** (Theorem 11: Syndrome-Free Dichotomy). *Every syndrome-free fault is either a spacetime logical fault or a spacetime stabilizer: if  $\text{IsSyndromeFree}(\delta, F)$  holds, then*

$$\text{IsSpacetimeLogicalFault}(\delta, F) \vee \text{IsSpacetimeStabilizer}(\delta, F).$$

Moreover, this dichotomy is exclusive.

*Proof.* We consider two cases based on whether  $\text{outcomePreserving}(F)$  holds.

If  $\text{outcomePreserving}(F)$  is true, then  $F$  satisfies both syndrome-freeness (by hypothesis) and outcome preservation, making it a spacetime stabilizer.

If  $\text{outcomePreserving}(F)$  is false, then  $F$  satisfies syndrome-freeness (by hypothesis) and outcome-changing (the negation of outcome-preserving), making it a spacetime logical fault.

For exclusivity, suppose  $F$  is both a logical fault and a stabilizer. Then  $F$  would be both outcome-changing (from the logical fault condition) and outcome-preserving (from the stabilizer condition). Since outcome-changing is defined as the negation of outcome-preserving, this yields a contradiction.  $\square$

This dichotomy has important operational consequences. Spacetime logical faults represent genuine threats to the encoded quantum information, as they modify either the measurement sign or apply non-trivial logical operators that cannot be detected through syndrome measurements. In contrast, spacetime stabilizers correspond to error patterns that, while physically present, have no net effect on the logical information or measurement outcomes.

### 1.32 Definition 12: SpacetimeFaultDistance

The spacetime fault-distance provides a fundamental measure of the robustness of quantum error correction schemes operating in both space and time. In fault-tolerant quantum computation, errors can occur not only on physical qubits during gate operations (space-faults) but also during measurements and state preparations (time-faults). The spacetime fault-distance quantifies the minimum number of such elementary errors required to cause an undetectable logical failure, making it a crucial parameter for understanding the error correction capabilities of a quantum code under realistic fault models.

This distance measure extends classical coding theory concepts to the quantum setting by accounting for the temporal aspects of quantum computation. Unlike static error correction codes, quantum fault-tolerant protocols must handle errors that accumulate over the entire computational process, including initialization, gate operations, and measurements. The spacetime perspective captures this dynamic nature by treating the entire computation as a spacetime lattice where errors can occur at any location and time.

**Definition** (Definition 12: Logical Fault Weights). Given a collection of detectors and an outcome-preserving predicate, the **set of logical fault weights** is

$$\{w \in \mathbb{N} \mid \exists F, F \text{ is a spacetime logical fault} \wedge |F| = w\}.$$

**Definition** (Definition 12: Spacetime Fault-Distance). The **spacetime fault-distance** is

$$d_{\text{spacetime}} = \inf (\text{logicalFaultWeights}(\text{detectors}, \text{outcomePreserving})),$$

where the infimum is taken over natural numbers, returning 0 for the empty set.

This definition captures the intuition that the distance represents the minimum "cost" of an undetectable error, where the cost is measured by the total weight of individual fault events. The weight  $|F|$  of a spacetime fault  $F$  counts each single-qubit Pauli error, measurement error, and initialization error as one unit.

**Theorem** (Theorem 12: Distance Upper Bound by Logical Fault Weight). *If  $F$  is a spacetime logical fault, then  $d_{\text{spacetime}} \leq |F|$ .*

*Proof.* Since  $F$  is a spacetime logical fault, we have  $|F| \in \text{logicalFaultWeights}$  (witnessed by the triple  $\langle F, h_{\log}, \text{rfl} \rangle$  where  $h_{\log}$  certifies that  $F$  is logical and  $\text{rfl}$  provides the weight equality). Therefore, by the definition of infimum,  $\inf(\text{logicalFaultWeights}) \leq |F|$ , which gives us  $d_{\text{spacetime}} \leq |F|$ .  $\square$

**Theorem** (Theorem 12: Zero Distance When No Logical Faults Exist). *If no spacetime logical faults exist, then  $d_{\text{spacetime}} = 0$ .*

*Proof.* We show that the set of logical fault weights is empty. For any weight  $w$ , the membership condition requires the existence of a spacetime logical fault  $F$  with weight  $w$ . However, by hypothesis, no spacetime logical faults exist, so no such  $F$  can be found for any  $w$ . Therefore,  $\text{logicalFaultWeights} = \emptyset$ . By definition of the spacetime fault-distance and the convention that the infimum of the empty set of natural numbers is 0, we conclude  $d_{\text{spacetime}} = 0$ .  $\square$

**Theorem** (Theorem 12: Positive Distance When Logical Faults Exist). *If the empty fault is outcome-preserving and there exists a spacetime logical fault, then  $d_{\text{spacetime}} > 0$ .*

*Proof.* First, we establish that  $0 \notin \text{logicalFaultWeights}$ . Suppose for contradiction that 0 belongs to this set, witnessed by some fault  $F$  with weight 0. By the positivity property of logical fault weights (which follows from the assumption that the empty fault is outcome-preserving), any logical fault must have positive weight, giving us a contradiction.

Next, from the existence hypothesis, there exists a spacetime logical fault  $F$ , which means  $|F| \in \text{logicalFaultWeights}$ , so the set is nonempty.

Now assume for contradiction that  $d_{\text{spacetime}} = 0$ . By the characterization of when the infimum of a set of natural numbers equals zero, either  $0 \in \text{logicalFaultWeights}$  (which we showed is false) or  $\text{logicalFaultWeights} = \emptyset$  (which contradicts nonemptiness). Therefore,  $d_{\text{spacetime}} > 0$ .  $\square$

**Theorem** (Theorem 12: Faults Below Distance Are Not Logical). *If  $|F| < d_{\text{spacetime}}$ , then  $F$  is not a spacetime logical fault.*

*Proof.* We proceed by contradiction. Assume that  $F$  is a spacetime logical fault. Then by the distance upper bound theorem, we have  $d_{\text{spacetime}} \leq |F|$ . This contradicts the hypothesis  $|F| < d_{\text{spacetime}}$ , completing the proof.  $\square$

**Theorem** (Theorem 12: Syndrome-Free Below Distance Implies Stabilizer). *If  $F$  is syndrome-free and  $|F| < d_{\text{spacetime}}$ , then  $F$  is a spacetime stabilizer.*

*Proof.* Since  $|F| < d_{\text{spacetime}}$ , the previous theorem establishes that  $F$  is not a spacetime logical fault. For a syndrome-free fault, there is a fundamental characterization: such a fault is a spacetime stabilizer if and only if it is not a spacetime logical fault. Since  $F$  satisfies both conditions (syndrome-free and not logical), we conclude that  $F$  is a spacetime stabilizer.  $\square$

The spacetime fault-distance also extends naturally to the gauging setting, where we consider fault-tolerant implementations of gauge-fixing measurements. This leads to the parallel development of gauging logical fault weights and gauging spacetime fault-distance, with analogous properties and bounds. The gauging distance measures the robustness of the gauge-fixing procedure itself, providing a complementary perspective on fault tolerance in gauge quantum error correction schemes.

### 1.33 Lemma 5: Spacetime Stabilizers

The spacetime stabilizer generators arise from the fundamental requirement that fault-tolerant quantum error correction must preserve the logical information while maintaining syndrome-free operation. In the context of the fault-tolerant gauging measurement procedure, we need to characterize all possible error patterns that leave no detectable trace in the syndrome measurements yet preserve the final logical outcome. These patterns form the stabilizer group of the spacetime code, analogous to how Pauli stabilizers generate the codespace in ordinary quantum error correction.

The key insight is that spacetime stabilizers decompose into time-localized generators based on the three-phase structure of the gauging procedure. Each generator consists of carefully orchestrated space-faults (Pauli errors) and time-faults (measurement errors) that cancel each other's effects on the detector measurements. The algebraic structure ensures that Pauli errors at consecutive time steps compose to the identity ( $P \cdot P = I$ ), while measurement faults on anticommuting checks create detector violations that cancel via the  $(-1) \times (-1) = +1$  rule in  $\mathbb{Z}/2\mathbb{Z}$ .

**Lemma** (Lemma 5: Spacetime Stabilizers). *Every listed generator fault pattern is a gauging stabilizer: it produces empty syndrome and preserves the logical measurement outcome.*

*Proof.* We establish that each type of listed generator satisfies both syndrome-freeness and gauging sign preservation.

**Space Stabilizer Generators (proven):** For generators with empty time-faults, both properties follow immediately. Any fault  $F$  with  $F.\text{timeFaults} = \emptyset$  is trivially syndrome-free since no detector measurements are affected. Sign preservation follows because the gauging sign flip formula sums over measurement faults, yielding zero when no measurement faults are present.

**Time-Propagating Generators (axiom):** Consider a generator with Pauli  $P$  at consecutive times  $t$  and  $t + 1$ , together with measurement faults on all checks anticommuting with  $P$  at the intermediate measurement round. Each detector spanning this time interval receives exactly two violations: one from the Pauli error and one from the corresponding measurement fault. Since violations are recorded in  $\mathbb{Z}/2\mathbb{Z}$ , we have  $1 + 1 = 0$ , so each detector reads as unviolated. The net Pauli effect is  $P \cdot P = I$ , preserving the logical state. *This case is stated as an axiom (`timePropagating_isGaugingStabilizer`) in the formalization, as the full detector-level cancellation argument was not formally verified.*

**Boundary Generators (axioms):** *Initialization*  $+X_e$ : The  $|0\rangle_e$  initialization fault flips the initialization detector for edge  $e$ . The  $X_e$  Pauli fault at gauging start time  $t_i$  creates a violation

in the same detector via the stabilizer measurement. These cancel:  $(-1) \times (-1) = +1$ . *Axiom: `initXe_isGaugingStabilizer`*. *Readout  $X_e$* : The  $X_e$  Pauli at ungauging time  $t_o$  flips the  $Z_e$  eigenvalue. The  $Z_e$  readout measurement fault compensates, effectively recording the correct eigenvalue. *Axiom: `readoutXe_isGaugingStabilizer`*.

**$Z_e + A_v$  Measurement Fault Generators (axiom)**: A  $Z_e$  Pauli error anticommutes with the Gauss law checks  $A_v$  for exactly the two vertices  $v \in e$  (since edges have precisely two endpoints). The corresponding  $A_v$  measurement faults create violations that cancel the Pauli-induced violations in all affected detectors. *Axiom: `ZeAvMeas_isGaugingStabilizer`*.

The algebraic foundation relies on eleven key properties, all **fully proven** in the formalization: self-inverse nature of all Pauli operators and checks ( $P^2 = I$ ), pairwise commutativity within each check family, the characterization that  $Z_e$  commutes with  $A_v$  if and only if  $v \notin e$ , and the edge structure ensuring exactly two anticommutation relationships per  $Z_e$ . These properties ensure that the measurement fault patterns precisely compensate for Pauli-induced detector violations while preserving the logical information encoded in the quantum state.

**Note on Completeness (axiom)**: The classification is complete in the sense that every gauging stabilizer decomposes as a  $\mathbb{Z}/2\mathbb{Z}$  combination of these listed generators. The proof uses time-ordered decomposition: generators at the earliest active time are isolated and canceled, with the process continuing inductively until only measurement faults remain, which must form detector measurement sets to maintain syndrome-freeness. *This completeness result is stated as an axiom (`spacetimeStabilizer_completeness`) in the formalization.*

**Status**: The main theorem (`listedGenerator_isGaugingStabilizer`) is proven by case analysis over generator types, reducing each case to one of the above results. The space stabilizer case and all algebraic properties (Parts I–III, VII in the Lean file) are fully verified. The four non-trivial generator cases (time-propagating, `init  $X_e$` , `readout  $X_e$` ,  `$Z_e + A_v$` ) and the completeness claim are axiomatized. Additionally, two downstream results used in Lemma 7 are axioms: `space_fault_cleaning` (space-fault concentration via time-propagating generators) and `syndromeFree_pureSpace_inCentralizer` (centralizer membership of cleaned faults).  $\square$

This result establishes the fundamental building blocks of the spacetime stabilizer group, providing the algebraic foundation for proving fault-tolerance of the gauging measurement procedure. The generators ensure that the quantum computation can tolerate errors up to the designed threshold while maintaining the integrity of both the encoded quantum information and the syndrome measurement data used for error detection.

### 1.34 Lemma 6: TimeFaultDistance

The time fault-distance is a fundamental parameter that quantifies the minimum cost of logical errors occurring purely through measurement faults in the time direction of the fault-tolerant gauging procedure. Understanding this distance is crucial for determining the error-correction threshold and the robustness of the protocol against temporal noise. In quantum error correction, temporal faults correspond to measurement errors that occur during the syndrome extraction process, as opposed to spatial faults that affect the data qubits themselves.

The key insight is that pure-time logical faults must satisfy stringent constraints due to the syndrome detection mechanism: they must remain undetectable by all detectors while still affecting the logical outcome. This tension between invisibility and logical action leads to a precise characterization of the minimum fault weight required.

**Definition** (Definition: Pure-Time Fault). A spacetime fault  $F$  is a **pure-time fault** if it has no space-faults, i.e.,  $F$  satisfies  $F.\text{isPureTime}$ .

**Definition** (Definition: Pure-Time Logical Fault). A spacetime fault  $F$  is a **pure-time gauging logical fault** if it is both a pure-time fault and a gauging logical fault:

$$\text{IsPureTimeLogicalFault}(F) \Leftrightarrow \text{IsPureTimeFault}(F) \wedge \text{IsGaugingLogicalFault}(F).$$

**Definition** (Definition: Time Fault-Distance). The **time fault-distance** is the infimum of the set of pure-time logical fault weights:

$$d_{\text{time}} = \inf \{w \in \mathbb{N} \mid \exists F, \text{IsPureTimeLogicalFault}(F) \wedge \text{weight}(F) = w\}.$$

To establish the time fault-distance, we construct an explicit witness fault that achieves the minimum weight. This fault corresponds to applying measurement errors to all  $A_v$  measurements for a fixed vertex  $v$  across all  $d$  rounds of Phase 2.

**Definition** (Definition: Gauss String Fault). The **Gauss string fault** for vertex  $v$  is the spacetime fault  $(\emptyset, \text{gaussMeasFaults}(v))$  with no space-faults and time-faults given by the  $A_v$  measurement string:

$$\text{gaussMeasFaults}(v) = \{\langle \text{phase2}(\text{gaussLaw}(v, r)) \rangle \mid r \in \text{Fin}(d)\}.$$

**Lemma** (Lemma 6: Time Fault-Distance). *When  $d$  is odd and the vertex set  $V$  is nonempty, the time fault-distance equals the number of deformed code rounds:*

$$d_{\text{time}} = d.$$

*Proof.* We establish both upper and lower bounds.

**Upper bound** ( $d_{\text{time}} \leq d$ ): We construct an explicit pure-time logical fault of weight  $d$ . Consider the Gauss string fault  $\text{gaussStringFault}(v)$  for any vertex  $v \in V$ . This fault has weight exactly  $d$  since it contains precisely  $d$  time-faults (one  $A_v$  measurement per round in Phase 2) and no space-faults.

We verify that this fault is syndrome-free by checking all detector types: *Phase 1 and Phase 3 detectors*: These involve measurements disjoint from the  $A_v$  measurements in our fault, so no violations occur. *Flux and deformed detectors*: These also involve measurements disjoint from Gauss measurements. *Gauss repeated detectors*: For the detector comparing  $A_w$  at consecutive rounds  $r$  and  $r'$ , if  $w = v$ , both measurements are faulted, giving flip parity  $1 + 1 = 0 \pmod{2}$  (no violation). If  $w \neq v$ , neither measurement is faulted, again giving parity 0.

For the logical condition, we compute the Gauss sign flip:

$$\text{gaussSignFlip}(\text{gaussStringFault}(v)) = \sum_{w \in V} \sum_{r \in \text{Fin}(d)} \llbracket A_w \text{ at round } r \text{ is faulted} \rrbracket \pmod{2}.$$

Only the terms with  $w = v$  contribute, giving a total of  $d \pmod{2} = 1$  (since  $d$  is odd). This non-zero sign flip confirms that the fault is logical.

**Lower bound** ( $d \leq d_{\text{time}}$ ): We prove that any pure-time logical fault must have weight at least  $d$ . Let  $F$  be such a fault. Since  $F$  is logical, it flips the gauging sign, so  $\text{gaussSignFlip}(F) = 1$ .

For any vertex  $v$ , define the *Gauss fault count* as the number of rounds where the  $A_v$  measurement is faulted. Due to syndrome-freeness, the Gauss repeated detectors impose strong constraints: if  $A_v$  is faulted at any round  $r$ , then by the constraint from consecutive detector pairs, it must be

faulted at round  $r + 1$  as well. By induction, this propagates to all rounds, so the Gauss fault count for each vertex is either 0 or  $d$ .

The sign flip condition becomes:

$$1 = \text{gaussSignFlip}(F) = \sum_{v \in V} \text{gaussFaultCount}(v, F) \pmod{2}.$$

Since each fault count is either 0 or  $d$ , and  $d$  is odd, each contributes either 0 or 1 to the sum modulo 2. For the sum to equal 1, an odd number of vertices must have fault count  $d$ . In particular, at least one vertex  $v$  satisfies  $\text{gaussFaultCount}(v, F) = d$ .

The weight of  $F$  satisfies:

$$\text{weight}(F) = |F.\text{timeFaults}| \geq \sum_{v \in V} \text{gaussFaultCount}(v, F) \geq d,$$

where the first inequality holds because each Gauss measurement contributes at most once to the total fault count (measurements for different vertices or different rounds are distinct), and the second follows from having at least one vertex with count  $d$ .

Combining the bounds gives  $d_{\text{time}} = d$ . □

This result reveals that the time fault-distance is determined entirely by the duration of the deformed code phase. The minimum-weight logical faults correspond precisely to corrupting all measurements of a single Gauss law  $A_v$  throughout Phase 2. This structure reflects the deep connection between the temporal fault-tolerance and the underlying gauge theory: logical errors manifest through consistent violations of local gauge constraints across time.

### 1.35 Lemma 7: SpaceTimeDecoupling

The independence of space and time components in quantum error correction is a fundamental principle that allows us to analyze logical faults separately along spatial and temporal dimensions. In fault-tolerant quantum computation, errors can occur both in space (affecting qubits directly) and in time (affecting measurement outcomes), but these effects can be decoupled under certain conditions. This separation is crucial for understanding the structure of logical operators and for proving distance bounds in topological quantum codes.

**Lemma** (Lemma 7: Space-Time Decoupling). *Any spacetime logical fault  $F$  in the fault-tolerant gauging measurement procedure can be decomposed into independent space and time components. Specifically, there exist faults  $F_S$ ,  $F_T$ , and a stabilizer  $S$  such that:*

1.  $F = (F_S \cdot F_T) \cdot S$  where composition is via symmetric difference,
2.  $F_S$  is pure-space (no time-faults) and concentrated at the gauging time  $t_i$ ,
3.  $F_T$  is pure-time (no space-faults) and syndrome-free,
4.  $S$  is a full gauging stabilizer (syndrome-free and outcome-preserving),
5. At least one of  $F_S$ ,  $F_T$  is nontrivial: either  $F_T$  flips the gauging sign or  $F_S$  applies a nontrivial logical operator to the code state.

Moreover, the space and time components affect different aspects independently:

- The gauging sign depends only on time-faults:  $\text{gaussSignFlip}(F) = \text{gaussSignFlip}(F_T)$
- The Pauli error depends only on space-faults:  $F.\text{pauliErrorAt}(t) = F_S.\text{pauliErrorAt}(t)$  for all  $t$

*Proof.* The proof proceeds by systematically cleaning the fault using spacetime stabilizers to separate space and time effects.

**Step 1: Space-fault cleaning (relies on axiom).** Any syndrome-free spacetime fault can be composed with a full gauging stabilizer to concentrate all space-faults at the gauging time  $t_i$ . From the space fault cleaning result (*axiom: space\_fault\_cleaning*), applied to  $F$  and its syndrome-freeness property, we obtain a stabilizer  $S_1$  such that for all times  $t \neq t_i$ ,  $(F \cdot S_1).\text{spaceFaultsAt}(t) = \emptyset$ . The composition  $F' := F \cdot S_1$  remains syndrome-free since composing syndrome-free faults preserves syndrome-freeness.

**Step 2: Boundary fault absorption.** For faults already concentrated at  $t_i$ , boundary faults can be trivially absorbed. We obtain a stabilizer  $S_2$  such that  $F'' := F' \cdot S_2$  maintains the space-fault concentration property. Again,  $F''$  remains syndrome-free.

**Step 3: Space-time decomposition.** Every spacetime fault naturally decomposes as  $F'' = F''.\text{spaceComponent} \cdot F''.\text{timeComponent}$ . Let  $F_S := F''.\text{spaceComponent}$ ,  $F_T := F''.\text{timeComponent}$ , and  $S := S_1 \cdot S_2$ . Since the composition of full gauging stabilizers is a full gauging stabilizer,  $S$  has the required properties.

We verify the decomposition  $F = (F_S \cdot F_T) \cdot S$ : Since  $F'' = F \cdot S$  and  $F'' = F_S \cdot F_T$ , we have  $F \cdot S = F_S \cdot F_T$ . By the involutive property of composition ( $F \cdot S \cdot S = F$ ), we obtain  $F = (F_S \cdot F_T) \cdot S$ .

**Step 4: Component properties.** By construction:

- $F_S$  is pure-space (contains only space-faults) and concentrated at  $t_i$
- $F_T$  is pure-time (contains only time-faults) and inherits syndrome-freeness from  $F''$
- $S$  is a full gauging stabilizer by composition properties

**Step 5: Independence of effects.** The gauging sign depends only on time-faults since it is computed as a sum over time-fault indicators. Therefore,  $\text{gaussSignFlip}(F) = \text{gaussSignFlip}(F_T)$ . Similarly, the Pauli error at any time depends only on space-faults, so  $F.\text{pauliErrorAt}(t) = F_S.\text{pauliErrorAt}(t)$ .

**Step 6: Nontriviality.** Since  $F$  is a logical fault, it does not preserve the full outcome. By the flip-or-preserve dichotomy, either  $F$  flips the gauging sign or preserves it. If  $F$  flips the sign, then by the independence property,  $F_T$  flips the sign. If  $F$  preserves the sign but changes the outcome, then the Pauli error is not in the stabilizer group, which by multiplicativity and the independence property implies that  $F_S$  applies a nontrivial logical operator. The centralizer membership of the cleaned pure-space fault follows from *axiom: syndromeFree\_pureSpace\_inCentralizer*.

**Axiom dependencies:** This proof relies on two axioms from Lemma 5: `space_fault_cleaning` (Step 1) and `syndromeFree_pureSpace_inCentralizer` (Step 6, centralizer membership).  $\square$

This decomposition is fundamental because it shows that logical faults in the fault-tolerant gauging procedure can be understood as combinations of purely spatial logical operators (affecting the quantum state) and purely temporal logical operators (affecting the classical measurement record). The independence of these effects allows for separate analysis of space and time error correction capabilities, which is essential for proving distance bounds and understanding the fault-tolerance threshold.

### 1.36 Theorem 2: FaultTolerantGaugingDistance

The fault-tolerant gauging procedure exhibits a remarkable property: its spacetime fault-distance exactly equals the distance of the original stabilizer code. This fundamental result establishes that the gauging measurement preserves the error-correcting capabilities of the underlying quantum error-correcting code, provided the procedure satisfies certain graph-theoretic and temporal constraints.

The key insight is that logical faults in the gauging procedure can be decomposed into space and time components, each subject to different distance bounds. Space faults correspond to errors on the original code qubits, while time faults arise from measurement errors during the gauging rounds. The interplay between these two types of faults, combined with the expansion properties of the underlying graph, determines the overall fault tolerance.

**Theorem** (Theorem 2: Fault-Tolerant Gauging Distance). *The spacetime fault-distance of the fault-tolerant gauging measurement procedure equals  $d$ , the distance of the original  $[[n, k, d]]$  stabilizer code, when the graph  $G$  has Cheeger constant  $h(G) \geq 1$  and the number of deformed code rounds satisfies  $t_o - t_i \geq d$ :*

$$d_{\text{spacetime}} = d.$$

*Proof.* We establish both directions of the equality through explicit constructions and case analysis.

**Upper bound** ( $d_{\text{spacetime}} \leq d$ ): We construct a space-fault witness by placing a minimum-weight logical operator  $P$  of the original code as a collection of space-faults at time  $t_i$ . For each qubit  $q$  in the support of  $P$ , we create a space fault  $(q, t_i, P.\text{xVec}(q), P.\text{zVec}(q))$ . This spacetime fault  $F = (\text{spaceFaultsOfPauliOp}(P, t_i), \emptyset)$  has no time-faults and is therefore syndrome-free.

The Pauli error at time  $t_i$  equals  $P$  by construction, so  $F$  is not outcome-preserving (since  $P$  is logical, not in the stabilizer group). Thus  $F$  is a full gauging logical fault with weight equal to  $\text{weight}(P) = d$ , giving  $d_{\text{spacetime}} \leq d$ .

**Lower bound** ( $d_{\text{spacetime}} \geq d$ ): Let  $F$  be any full gauging logical fault. Using the space-time decomposition, we write  $F = F_S \cdot F_T \cdot S$  where  $F_S$  is pure-space,  $F_T$  is pure-time and syndrome-free, and  $S$  is a full gauging stabilizer.

We consider two cases:

*Case (a):  $F_T$  flips the gauging sign.* The sign-flip property implies  $d$  must be odd (if  $d$  were even, the all-or-none property of time faults would force the total sign flip to be zero). The time component provides a logical fault of weight  $\geq d$  by the time fault distance bound.

*Case (b):  $F_S$  yields a nontrivial deformed code logical.* The Pauli error  $\text{pauliErrorAt}(F, t_i) = \text{pauliErrorAt}(F_S, t_i) \cdot \text{pauliErrorAt}(S, t_i)$  is a logical operator of the deformed code (in the centralizer but not in the stabilizer group). Since the deformed code distance satisfies  $d' \geq d$  when  $h(G) \geq 1$ , and the Pauli error weight is bounded by the space weight, we obtain  $d \leq d' \leq \text{weight}(\text{pauliErrorAt}(F, t_i)) \leq \text{spaceWeight}(F) \leq \text{weight}(F)$ .

In both cases,  $\text{weight}(F) \geq d$ , so  $d_{\text{spacetime}} \geq d$ .

Combining both bounds yields  $d_{\text{spacetime}} = d$ .

**Axiom dependencies:** This proof uses the space-time decomposition (Lemma 7), which transitively depends on the axioms `space_fault_cleaning` and `syndromeFree_pureSpace_inCentralizer` from Lemma 5. □

This theorem has several important implications. First, it shows that the gauging procedure achieves optimal fault tolerance: no syndrome-free fault of weight less than  $d$  can cause a logical error. Second, it establishes that the expansion condition  $h(G) \geq 1$  is sufficient to preserve the distance, connecting the algebraic properties of the code to the geometric properties of the measurement graph. Finally, the equality  $d_{\text{spacetime}} = d$  demonstrates that the additional complexity

of the spacetime measurement does not degrade the fundamental error-correcting capabilities of the original code.

### 1.37 Remark 15: FluxCheckMeasurementFrequency

In quantum error correction, a fundamental challenge arises when implementing fault-tolerant quantum codes: how frequently must syndrome measurements be performed to maintain the error correction guarantees? For topological codes with both vertex checks ( $A_v$ ) and flux checks ( $B_p$ ), one might naively assume that all stabilizers must be measured with equal frequency. However, as we explore in this remark, the flux measurements have a special property that allows for much more flexible measurement schedules.

The key insight is that flux operators  $B_p = \prod_{e \in p} Z_e$  can be inferred indirectly from other measurements rather than being directly observed. This observation has profound implications for practical implementations, where reducing measurement overhead is crucial for maintaining coherence times and achieving fault tolerance thresholds.

*Remark* (Remark 15: Flux Check Measurement Frequency). The fault-tolerance proof (Theorem 2) remains valid even when flux checks  $B_p$  are measured much less frequently than every round, or not directly measured at all. The fundamental reason is threefold:

1. Flux values can be inferred from qubit initialization and final readout measurements
2. The distance bound requires only that  $B_p$  operators are stabilizers, not that they are measured
3. Time-fault analysis shows that  $A_v$  measurements are the bottleneck for fault tolerance

However, this flexibility comes with important caveats: detector cells become large when measurements are infrequent, which prevents achieving optimal thresholds against continuous noise.

The mathematical foundation for this flexibility rests on several key results that we now establish.

**Theorem** (Theorem: Flux Inferred from Init Readout). *The flux init detector ( $B_p^{ti}$ ) captures the relationship between edge initialization and the first flux measurement. For every edge  $e \in p$  belonging to the cycle of plaquette  $p$ , the edge initialization event for  $e$  is contained in the measurements of the flux init detector.*

*Proof.* Let  $e$  be an edge with  $e \in \text{cycles}(p)$ . The result follows directly from the construction of the flux init detector, which by definition includes all edge initialization measurements for edges in the plaquette cycle. Since qubits are initialized in the  $|0\rangle$  state, which is a  $+1$  eigenstate of  $Z_e$ , the flux value  $B_p = \prod_{e \in p} Z_e$  at initialization is fully determined by these initialization measurements.  $\square$

**Theorem** (Theorem: Flux Inferred from Readout). *The flux ungauge detector ( $B_p^{to}$ ) captures the relationship between the last flux measurement and individual  $Z_e$  readouts for  $e \in p$ . For every edge  $e$  in the cycle of plaquette  $p$ , the Phase 3 edge- $Z$  readout measurement for  $e$  is contained in the measurements of the flux ungauge detector.*

*Proof.* Let  $e$  be an edge with  $e \in \text{cycles}(p)$ . The flux ungauge detector for plaquette  $p$  at the final round contains the Phase 3 edge- $Z$  measurement for each edge in the cycle by construction. Since  $B_p = \prod_{e \in p} Z_e$ , the flux value at the end of the computation can be reconstructed from these individual  $Z_e$  measurements.  $\square$

The distance properties of the code depend only on algebraic relationships, not on measurement frequencies:

**Theorem** (Theorem: Space Distance Independent of Flux Measurements). *The distance bound from Lemma 3 requires only that  $B_p$  operators are elements of the deformed stabilizer group. No measurement of  $B_p$  is needed for  $d^* \geq d$ . Given sufficient expansion  $h(G) \geq 1$ , we have*

$$d(\text{original code}) \leq d(\text{deformed code}).$$

*Proof.* The proof relies on the space distance theorem for deformed codes. When the graph  $G$  has sufficient expansion, the deformed code distance is at least the original code distance. Crucially, this result uses only:

1. Algebraic properties of flux operators as stabilizers
2. Graph connectivity properties
3. Coboundary exactness conditions
4. The Cheeger expansion bound

No flux measurements are required in the analysis—only the mathematical fact that  $B_p$  operators commute with all other stabilizers and preserve the code space.  $\square$

For time-fault analysis, the situation is even more striking:

**Theorem** (Theorem: Time Fault Bottleneck Is Gauss). *The time-fault distance is determined by  $A_v$  measurement strings, not  $B_p$ . The canonical minimum-weight pure-time logical fault is a single  $A_v$  string of weight  $d$ . For any vertex  $v$  and odd  $d$ :*

$$d_{\text{time}} \leq d.$$

*Proof.* This follows by explicitly constructing the Gauss string fault for vertex  $v$ . This fault affects exactly  $d$  consecutive  $A_v$  measurements for vertex  $v$ , creating a syndrome-free pattern that flips the gauging sign. The construction requires no flux measurements—the entire fault pattern involves only vertex checks  $A_v$ .  $\square$

**Theorem** (Theorem: Flux Fault Preserves Sign).  *$B_p$  measurement faults cannot flip the gauging sign  $\sigma$ . If every time fault in a fault pattern  $F$  is a flux measurement, then  $F$  preserves the gauging sign.*

*Proof.* The gauging sign is defined as  $\sigma = \sum_v \varepsilon_v$ , where  $\varepsilon_v$  indicates whether vertex  $v$  has an odd number of  $A_v$  measurement faults. Consider a fault  $F$  whose time faults contain only flux measurements of the form  $\langle \text{phase2}(\text{flux } p \ r) \rangle$ .

We must show that  $F$  contributes 0 to the sign sum. The sign flip is computed as:

$$\sum_{v \in V} \sum_{r \in \{0, \dots, d-1\}} \mathbf{1}[\langle \text{phase2}(\text{gaussLaw } v \ r) \rangle \in F.\text{timeFaults}]$$

For each term in this double sum, suppose for contradiction that the indicator equals 1. Then we have  $\langle \text{phase2}(\text{gaussLaw } v \ r) \rangle = \langle \text{phase2}(\text{flux } p \ r') \rangle$  for some plaquette  $p$  and round  $r'$ . Since the `phase2` constructor is injective, this implies `gaussLaw`  $v \ r = \text{flux } p \ r'$ . However, `gaussLaw` and `flux` are distinct constructors, giving a contradiction.

Therefore, every indicator in the sum equals 0, so the total sign contribution is 0.  $\square$

The practical implications become clear when comparing detector sizes:

**Theorem** (Theorem: Boundary Larger Than Repeated). *When flux measurements are infrequent, detector cells become large. If a plaquette cycle contains at least 2 edges, then boundary detectors (spanning from initialization to final readout) contain strictly more measurements than repeated detectors:*

$$|\text{repeated detector measurements}| < |\text{boundary detector measurements}|.$$

*Specifically,  $2 < |p| + 1$  when  $|p| \geq 2$ .*

*Proof.* Repeated flux detectors contain exactly 2 measurements (two consecutive  $B_p$  measurements). The flux init detector contains  $|p| + 1$  measurements: one for each edge initialization in the cycle, plus the first flux measurement. When  $|p| \geq 2$ , we have  $|p| + 1 \geq 3 > 2$ , establishing the inequality.  $\square$

This analysis reveals a fundamental trade-off in quantum error correction: while flux measurements can be performed less frequently without compromising the theoretical fault tolerance guarantees, doing so creates larger detection cells that make the system more vulnerable to correlated noise and reduce the practical threshold for achieving fault tolerance. For small, fixed code instances, this approach may nonetheless be advantageous due to reduced measurement overhead.

### 1.38 Remark 16: BoundaryRoundsOverkill

The conventional approach to fault-tolerant quantum error correction often employs a conservative strategy when implementing measurements that bridge different phases of computation. In the context of gauging procedures for quantum stabilizer codes, this manifests as using  $d$  rounds of standard error correction both before and after the central gauging measurement, where  $d$  is the distance of the underlying code. While this choice facilitates clean theoretical analysis, it represents a significant overestimate of what is actually required in practice.

The motivation for  $d$  boundary rounds stems from ensuring that any combined error process involving the gauging measurement and the initial or final boundaries has weight exceeding  $d$ . This conservative approach enables a straightforward proof of fault-distance bounds, but closer examination reveals that the essential fault-tolerance properties depend primarily on the structure of the central gauging phase, with boundary phases playing only a supporting role.

*Remark* (Remark 16: Boundary Rounds Overkill). The  $d$  rounds of error correction in the original code before and after the gauging measurement (Phases 1 and 3) are conservative and often unnecessary in practice. The critical fault-tolerance properties are determined by the Phase 2 structure, with boundary phases contributing only minimal coverage requirements.

**Key observations:**

1. The all-or-none property (each vertex has either 0 or  $d$  Gauss measurement faults) is enforced entirely by Phase 2 repeated detectors
2. Boundary detectors connecting phases use exactly one measurement from each adjacent phase
3. Space-faults at the gauging time are detected by deformed code checks in Phase 2, independent of boundary rounds
4. A single boundary round ( $d_{\text{boundary}} = 1$ ) would suffice for detector coverage, though  $d$  rounds enable cleaner theoretical analysis

The practical implications are significant: when the gauging measurement is embedded within a larger fault-tolerant computation, the surrounding operations naturally provide boundaries for spacetime fault analysis. In such contexts, a constant number of boundary rounds—potentially much smaller than  $d$ —may suffice, leading to substantial improvements in circuit depth and threshold performance. The optimal choice depends on the specific computational context and represents an important area for optimization in practical quantum error correction implementations.

### 1.39 Corollary 1: WorstCaseOverhead

The analysis of quantum error correction overhead requires careful consideration of resource scaling as logical operator weight increases. For fault-tolerant implementations of quantum low-density parity-check (qLDPC) codes, understanding how auxiliary qubit requirements scale with the weight of measured logical operators is crucial for practical quantum computing. This corollary establishes that the worst-case overhead scales as  $O(W \log^2 W)$  for a logical operator of weight  $W$ , providing an efficient scaling that is independent of the total code size  $n$ .

The key insight is that by constructing appropriate expander graphs with bounded degree and using techniques from graph theory and algebraic topology, we can achieve this favorable scaling while preserving all essential code properties. The overhead arises from edge qubits introduced in the gauging procedure, but the logarithmic factors ensure the construction remains practical even for large logical operators.

**Corollary** (Corollary 1: Worst-Case Overhead). *For a qLDPC stabilizer code with a Pauli logical operator  $L$  of weight  $W$ , there exists a fault-tolerant gauging measurement procedure with total qubit overhead  $O(W \log^2 W)$ , where the implicit constant depends on the code parameters (check weight bound  $w$  and qubit participation bound  $c$ ) but not on  $W$  or the code size  $n$ .*

*More precisely, there exists a choice of constant-degree expander graph  $G$  on  $W$  vertices such that:*

1. *All construction desiderata are satisfied with bounded-weight checks and preserved distance  $d^* \geq d$*
2. *The number of additional edge qubits is at most  $\frac{\Delta \cdot (K \cdot (\log_2 W)^2 / c + 1) \cdot W}{2}$*
3. *Gauss law checks have weight at most  $1 + \Delta$  and flux checks have weight at most 4*
4. *The deformed code maintains the LDPC property*

*where  $\Delta$  is the maximum graph degree,  $K$  is the Freedman-Hastings constant, and  $c > 0$  is the per-layer cycle-degree bound.*

*Proof.* The proof proceeds by constructing an appropriate expander graph and analyzing the resulting overhead through several key steps.

**Step 1: Graph Construction.** We construct a constant-degree  $\Delta$  connected graph  $G$  on  $|V| = W$  vertices (corresponding to the support of logical operator  $L$ ) with sufficient expansion  $h(G) \geq 1$ . The graph satisfies the  $Z$ -support matching condition and admits a low-weight cycle basis with cycles of weight at most 4. The cycle rank property  $|C| + |V| = |E| + 1$  ensures the topological structure needed for gauging.

**Step 2: Layer Bound via Decongestion.** By the König coloring theorem combined with the Freedman-Hastings bound, we can partition the cycle set into at most  $M \leq K \cdot (\log_2 W)^2$

layers, where each layer satisfies a cycle-degree bound determined by parameter  $c$ . This gives  $R \leq K \cdot (\log_2 W)^2/c$  effective layers.

**Step 3: Edge Overhead Analysis.** The constant degree property ensures  $2|E| \leq \Delta \cdot |V| = \Delta \cdot W$ . Combined with the layer bound, the total number of edge qubits satisfies:

$$|E| \leq \frac{\Delta \cdot (R+1) \cdot W}{2} \leq \frac{\Delta \cdot (K \cdot (\log_2 W)^2/c + 1) \cdot W}{2}$$

This establishes the  $O(W \log^2 W)$  scaling.

**Step 4: Desiderata Verification.** The construction satisfies all required properties: *Bounded checks*: Gauss law checks have weight  $\leq 1 + \Delta$  and flux checks have weight  $\leq 4$  by the low-weight cycle basis *Distance preservation*: The sufficient expansion condition ensures  $d^* \geq d$  via the boundary expansion property *LDPC property*: Both check weight bounds and qubit degree bounds remain constant, independent of  $W$  and  $n$

**Step 5: Independence from Code Size.** The overhead bound depends only on  $W$ ,  $\Delta$ ,  $K$ , and  $c$ . Since these parameters are determined by the local structure of the code (check weights, qubit degrees) and the expansion properties, the constant is independent of the total code size  $n$ .

The multiplicative constant  $\Delta \cdot (K/c + 1)$  depends on code parameters but not on the logical operator weight  $W$  or code size  $n$ , completing the proof of the worst-case overhead bound.  $\square$

This result has significant implications for the practical implementation of fault-tolerant quantum computation with qLDPC codes. The  $O(W \log^2 W)$  scaling means that even for logical operators with substantial support, the auxiliary qubit overhead remains manageable. Moreover, the independence from total code size  $n$  allows this construction to scale favorably as quantum computers grow larger, since the overhead for measuring any particular logical operator depends only on its local structure, not the global code size.

## 1.40 Remark 17: HypergraphGeneralization

In the study of quantum error correction and topological stabilizer codes, one often encounters systems where the underlying structure is more complex than simple graphs. The natural generalization from graphs to hypergraphs opens up new possibilities for code construction and measurement protocols. Where graph-based codes associate qubits with vertices and edges, hypergraph-based codes can accommodate more intricate connectivity patterns where each auxiliary qubit corresponds to a hyperedge that can connect any number of vertices.

This generalization preserves the fundamental algebraic structure of the gauging framework while extending its scope. The key insight is that Gauss's law operators maintain their essential properties—mutual commutativity and the characterization of stabilizer constraints through kernel conditions—even in this more general setting.

*Remark* (Remark 17: Hypergraph Generalization). The gauging measurement procedure can be generalized by replacing the graph  $G$  with a hypergraph  $H = (V, E)$ , where each hyperedge  $e \in E$  is a nonempty subset of vertices. One introduces one auxiliary qubit per hyperedge, defines Gauss's law operators  $A_v = X_v \prod_{e \ni v} X_e$ , and a boundary map  $\partial : \mathbb{Z}_2^E \rightarrow \mathbb{Z}_2^V$ . The fundamental result is that  $\gamma \in \ker(\partial)$  if and only if the corresponding pure- $Z$  hyperedge operator commutes with all  $A_v$ .

The hypergraph structure requires careful definition of the boundary operators. For a hypergraph with vertex set  $V$  and hyperedge set  $E$ , we define the boundary map  $\partial : \mathbb{Z}_2^E \rightarrow \mathbb{Z}_2^V$  by

$$(\partial\gamma)_v = \sum_{e \in E} \begin{cases} \gamma_e & \text{if } v \text{ is incident to } e, \\ 0 & \text{otherwise.} \end{cases}$$

The coboundary map  $\delta : \mathbb{Z}_2^V \rightarrow \mathbb{Z}_2^E$  is the transpose of  $\partial$ , satisfying

$$(\delta f)_e = \sum_{v \in V} \begin{cases} f_v & \text{if } v \text{ is incident to } e, \\ 0 & \text{otherwise.} \end{cases}$$

A crucial property is that  $\delta$  and  $\partial$  are indeed transposes, meaning

$$\sum_{e \in E} (\delta f)_e \cdot \gamma_e = \sum_{v \in V} f_v \cdot (\partial \gamma)_v$$

for all  $f \in \mathbb{Z}_2^V$  and  $\gamma \in \mathbb{Z}_2^E$ .

The hypergraph Gauss's law operators are defined as  $A_v = X_v \prod_{e \ni v} X_e$ , where the product runs over all hyperedges incident to vertex  $v$ . These operators maintain the essential property that they are pure  $X$ -type (having no  $Z$  components) and mutually commute:  $[A_v, A_w] = 0$  for all vertices  $v, w$ .

The central result characterizing the kernel of the boundary map states that for any edge vector  $\gamma \in \mathbb{Z}_2^E$ , we have  $\gamma \in \ker(\partial)$  if and only if the corresponding pure- $Z$  hyperedge operator  $Z(\gamma) = \prod_{e: \gamma_e=1} Z_e$  commutes with all Gauss operators  $A_v$ . This equivalence provides the algebraic foundation for understanding which configurations of auxiliary qubits lead to valid stabilizer constraints.

The hypergraph framework reduces to the standard graph case when each hyperedge contains exactly two vertices, recovering all the familiar properties of CSS codes and topological stabilizer codes. This generalization thus provides a unified perspective on a broad class of quantum error-correcting codes while maintaining the essential algebraic structure that makes such codes tractable.

### 1.41 Remark 18: RelationToLatticeSurgery

The gauging measurement procedure, introduced earlier for quantum error correction, provides a powerful framework for entangling and disentangling separate quantum codes. We now demonstrate that this general framework encompasses lattice surgery as a special case, while also extending to long-range operations and LDPC code generalizations that go beyond traditional surface codes.

Lattice surgery is a fundamental technique in surface code quantum computation, allowing the merging and splitting of logical qubits by measuring Pauli operators on auxiliary qubits that connect separate code patches. The key insight of this remark is that lattice surgery can be understood as gauging on specific graph structures: ladder graphs for adjacent patches, grid graphs for distant patches, and bridge graphs for general LDPC codes.

*Remark* (Remark 18: Relation to Lattice Surgery). The gauging measurement procedure generalizes surface code lattice surgery through three distinct graph constructions:

1. **Ladder graphs** for standard lattice surgery between adjacent surface code patches
2. **Grid graphs** for long-range lattice surgery between distant patches
3. **Bridge graphs** for LDPC code generalization beyond surface codes

Each construction preserves the essential properties needed for fault-tolerant quantum computation while extending the scope of lattice surgery operations.

The ladder graph construction directly models traditional lattice surgery. Consider two surface code patches, each with  $n$  boundary qubits that need to be connected. The ladder graph provides the minimal structure needed for this connection.

**Definition** (Definition: Ladder Adjacency). The adjacency relation for the ladder graph on vertex set  $\mathbb{B} \times \times \leq \times \text{Fin}(n)$ . Two vertices  $p = (b_1, i)$  and  $q = (b_2, j)$  are adjacent if  $p \neq q$  and either:

1. **Rung condition:**  $b_1 \neq b_2$  and  $i = j$  (same position, different boundaries)
2. **Rail condition:**  $b_1 = b_2$  and  $|i - j| = 1$  (same boundary, consecutive positions)

This adjacency naturally creates a ladder structure where rungs connect corresponding qubits on opposite boundaries, while rails connect adjacent qubits within each boundary.

**Theorem** (Theorem: Ladder Graph Properties). *For  $n \geq 1$ , the ladder graph satisfies:*

1. *Connectivity: any two vertices can be reached by a path*
2. *Vertex count:  $|\mathbb{B} \times \times \leq \times \text{Fin}(n)| = 2n$*
3. *Bounded degree: each vertex has degree at most 3*
4. *Linear edge count: at most  $3n$  edges total*

*Proof. Connectivity:* We show every vertex  $(b, k)$  is reachable from  $(\text{false}, 0)$ . By induction on  $k$ , the rail edges connect  $(\text{false}, m)$  to  $(\text{false}, m + 1)$  for  $m < n - 1$ . This establishes reachability within the first boundary. The rung edges then connect each  $(\text{false}, k)$  to  $(\text{true}, k)$ , completing connectivity.

**Vertex count:** By cardinality of product types,  $|\mathbb{B} \times \times \leq| \cdot |\text{Fin}(n)| = 2 \cdot n = 2n$ .

**Bounded degree:** Each vertex  $v = (b, k)$  has at most three neighbors: the rung partner  $(\neg b, k)$ , and up to two rail neighbors  $(b, k \pm 1)$  (if they exist). Hence degree  $\leq 3$ .

**Linear edge count:** By the handshaking lemma,  $2|E| = \sum_v \deg(v) \leq 2n \cdot 3 = 6n$ , so  $|E| \leq 3n$ .  $\square$

The key insight is that gauging on the ladder graph exactly reproduces lattice surgery. The Gauss law operators  $A_v$  become vertex stabilizers (pure  $X$ -type), flux operators become face stabilizers, and the deformed check operators correspond to boundary stabilizers. The logical operator  $L = \prod_v A_v$  represents the joint logical  $X$  operator of the merged patch.

For distant surface code patches separated by distance  $D$ , we require a different construction that introduces auxiliary qubits to bridge the gap.

**Definition** (Definition: Grid Graph). The grid graph on vertex set  $\text{Fin}(n) \times \text{Fin}(D + 2)$  where:

- Column 0 contains boundary qubits from the first patch
- Column  $D + 1$  contains boundary qubits from the second patch
- Columns  $1, \dots, D$  contain auxiliary qubits

Adjacency follows a standard grid pattern: vertices are connected horizontally (same row, adjacent columns) or vertically (same column, adjacent rows).

**Theorem** (Theorem: Long-Range Surgery Overhead). *For boundary sets of size  $n \geq 1$  separated by distance  $D$ , the grid graph requires:*

1. *Total qubits:  $n(D + 2) = 2n + nD$*
2. *Overhead:  $n$  auxiliary qubits per unit distance*

3. *Connectivity: the graph remains connected for all  $n \geq 1$*

4. *Bounded degree: each vertex has degree at most 4*

*Proof. Qubit count:* The vertex set  $\text{Fin}(n) \times \text{Fin}(D+2)$  has cardinality  $n(D+2)$ . Decomposing:  $n(D+2) = n \cdot 2 + n \cdot D = 2n + nD$ , where  $2n$  are boundary qubits and  $nD$  are auxiliaries.

**Connectivity:** We show every vertex  $(r, c)$  is reachable from  $(0, 0)$ . First reach  $(0, c)$  by horizontal moves, then reach  $(r, c)$  by vertical moves. The composition gives a path to any vertex.

**Bounded degree:** Each vertex has at most four neighbors (up, down, left, right in the grid), so degree  $\leq 4$ .  $\square$

The grid construction enables lattice surgery between distant patches with overhead linear in the separation distance. This generalizes beyond surface codes to arbitrary distance requirements.

For the most general case, we consider LDPC codes that may not have geometric structure. Here we introduce the bridge graph construction.

**Definition** (Definition: Bridge Graph). Given a graph  $G$  on vertex set  $V$  and a bridge set  $\text{bridges} \subseteq V$ , the bridge graph has:

- Vertex set:  $V \oplus V$  (two disjoint copies of  $V$ )
- Adjacency: vertices are adjacent if they are adjacent within the same copy of  $G$ , or if they are corresponding vertices in different copies with the vertex in the bridge set

**Theorem** (Theorem: LDPC Lattice Surgery Generalization). *If  $G$  is connected with nonempty bridge set, and  $G$  satisfies LDPC desiderata (short paths with bound  $D$  and low-weight cycles with bound  $W$ ), then:*

1. *The bridge graph is connected with  $2|V|$  vertices*
2. *Short paths within each copy are preserved (bound  $D$ )*
3. *Low-weight cycle structure is preserved (bound  $W$ )*
4. *The construction works regardless of Cheeger expansion properties*

*Proof. Connectivity:* Let  $b$  be any vertex in the bridge set. Every vertex  $\text{inl}(v)$  in copy 1 is reachable from  $\text{inl}(b)$  using the connectivity of  $G$ . Every vertex  $\text{inr}(v)$  in copy 2 is reachable by first crossing the bridge from  $\text{inl}(b)$  to  $\text{inr}(b)$ , then using connectivity within copy 2.

**Short paths preservation:** For vertices  $u, v$  in the same copy, we have

$$\text{dist}_{\text{BridgeGraph}}(\text{inl}(u), \text{inl}(v)) \leq \text{dist}_G(u, v) \leq D$$

where the first inequality holds because embeddings are distance-preserving, and the second follows from the short paths property of  $G$ .

**Low-weight cycles:** Cycles within each copy inherit the weight bound  $W$  from the original graph  $G$ .

**Cheeger independence:** Unlike traditional LDPC requirements, the bridge construction succeeds based only on connectivity and local properties, not global expansion.  $\square$

This demonstrates a remarkable fact: the gauging framework enables lattice surgery operations on LDPC codes without requiring the strong expansion properties typically needed for LDPC code performance. The bridge construction provides a path around the usual Cheeger constant constraints.

The three constructions—ladder, grid, and bridge graphs—show that lattice surgery emerges naturally from the gauging framework. This unification reveals lattice surgery as a special case of a more general measurement-based approach to quantum code manipulation, opening possibilities for fault-tolerant operations on broader classes of quantum error-correcting codes.

## 1.42 Remark 19: BivariateBicycleCodeNotation

Bivariate Bicycle (BB) codes represent a powerful class of quantum error-correcting codes that exploit the algebraic structure of cyclic permutation matrices and their tensor products. These codes operate on  $2\ell m$  qubits and are constructed using polynomial algebra over finite fields, providing a systematic approach to designing CSS stabilizer codes with favorable properties.

The construction relies on the group algebra  $\mathbb{F}_2[x, y]/(x^\ell - 1, y^m - 1)$ , which naturally encodes the cyclic structure of the underlying permutation matrices. By representing X-type and Z-type parity checks as polynomials in this algebra, we obtain a framework where the CSS condition  $AB^T = BA^T$  ensures that all stabilizer generators commute, forming a valid stabilizer code.

*Remark* (Remark 19: Bivariate Bicycle Code Notation). The BB code construction uses the monomial group  $M = \mathbb{Z}_\ell \times \mathbb{Z}_m$  to represent monomials  $x^a y^b$  with  $a \in \{0, \dots, \ell - 1\}$  and  $b \in \{0, \dots, m - 1\}$ , where the additive group structure corresponds to monomial multiplication. The group algebra  $\mathbb{F}_2[x, y]/(x^\ell - 1, y^m - 1)$  consists of functions  $M \rightarrow \mathbb{Z}/2\mathbb{Z}$ , where the value at  $(a, b)$  gives the coefficient of  $x^a y^b$ .

For polynomials  $A, B$  in this algebra, the X-type check indexed by  $\alpha \in M$  is defined as

$$\text{check}(\alpha, X) = X(\text{shift}_\alpha A, \text{shift}_\alpha B),$$

where  $(\text{shift}_\alpha p)(\gamma) = p(\gamma - \alpha)$  represents left-shifting by monomial  $\alpha$ . Similarly, the Z-type check indexed by  $\beta \in M$  is

$$\text{check}(\beta, Z) = Z(\text{shift}_\beta B^T, \text{shift}_\beta A^T),$$

where the transpose operation satisfies  $A^T(\gamma) = A(-\gamma)$ , corresponding to the substitution  $x \rightarrow x^{-1}, y \rightarrow y^{-1}$ .

The resulting BB code operates on  $2\ell m$  qubits (split into left and right registers) with  $2\ell m$  checks, forming a CSS stabilizer code when the polynomials  $A$  and  $B$  satisfy the CSS condition  $AB^T = BA^T$  in the group algebra. This condition ensures that all X and Z checks mutually commute, with X checks automatically commuting among themselves and Z checks automatically commuting among themselves due to their pure types.

This algebraic framework provides significant computational advantages, as convolution operations in the group algebra correspond directly to matrix multiplication of the associated circulant-like parity check matrices. The systematic construction allows for explicit analysis of code parameters and facilitates the design of codes with specific distance properties through careful choice of the defining polynomials  $A$  and  $B$ .

### 1.43 Remark 20: GrossCodeDefinition

The Gross code represents a significant example in the family of bivariate bicycle codes, demonstrating how careful polynomial construction can yield quantum error-correcting codes with specific logical operator structures. This code achieves a balance between code parameters and geometric constraints that makes it particularly suitable for studying the relationship between algebraic construction and topological properties in quantum codes.

*Remark* (Remark 20: Gross Code Definition). The **Gross code** is a  $[[144, 12, 12]]$  bivariate bicycle (BB) code with parameters  $\ell = 12$ ,  $m = 6$ . The polynomials defining the code are  $A = x^3 + y^2 + y$  and  $B = y^3 + x^2 + x$  in  $\mathbb{F}_2[x, y]/(x^{12} - 1, y^6 - 1)$ .

The construction begins with the fundamental algebraic structures. The monomial group is  $\mathbb{Z}_{12} \times \mathbb{Z}_6$ , and the corresponding group algebra is  $\mathbb{F}_2[x, y]/(x^{12} - 1, y^6 - 1)$ . The qubit system consists of left ( $L$ ) and right ( $R$ ) qubits indexed by elements of  $\mathbb{Z}_{12} \times \mathbb{Z}_6$ , giving a total of 144 physical qubits.

The defining polynomials satisfy the crucial commutativity condition. For all  $\alpha, \beta \in \mathbb{Z}_{12} \times \mathbb{Z}_6$ , the X-check  $\text{bbCheckX}(A, B, \alpha)$  and Z-check  $\text{bbCheckZ}(A, B, \beta)$  commute as Pauli operators. This commutativity extends to all pairs of checks, ensuring the stabilizer group is well-defined.

The logical operators are constructed from specific polynomials:

$$f = 1 + x + x^2 + x^3 + x^6 + x^7 + x^8 + x^9 + (x + x^5 + x^7 + x^{11})y^3 \quad (11)$$

$$g = x + x^2y + (1 + x)y^2 + x^2y^3 + y^4 \quad (12)$$

$$h = 1 + (1 + x)y + y^2 + (1 + x)y^3 \quad (13)$$

These polynomials determine the logical operators:

$$\bar{X}_\alpha = X(\alpha \cdot f, 0) \quad (14)$$

$$\bar{X}'_\beta = X(\beta \cdot g, \beta \cdot h) \quad (15)$$

$$\bar{Z}_\beta = Z(\beta \cdot h^T, \beta \cdot g^T) \quad (16)$$

$$\bar{Z}'_\alpha = Z(0, \alpha \cdot f^T) \quad (17)$$

A key property of these logical operators is their type purity: all X-type logicals have zero Z-support, and all Z-type logicals have zero X-support. Furthermore, the operators  $\bar{X}_\alpha$  act exclusively on left qubits, while  $\bar{Z}'_\alpha$  act exclusively on right qubits.

The weight properties reveal important geometric constraints. The polynomial  $f$  has exactly 12 nonzero coefficients, giving each logical operator  $\bar{X}_\alpha$  weight 12. The defining polynomials  $A$  and  $B$  each have support of size 3, which limits the expansion properties of the Tanner graph.

All logical operators are self-inverse and commute with every stabilizer check, confirming their membership in the centralizer. The X-checks have weight 6, and the code has exactly 144 checks, matching the number of qubits.

The Gross code demonstrates an interesting limitation: since  $\bar{X}_\alpha$  acts only on left qubits and each left qubit participates in at most 3 X-type checks (determined by  $|\text{supp}(A)| = 3$ ), the Tanner subgraph on the support of any logical X operator has limited expansion. This constraint illustrates the fundamental tension between achieving low-weight logical operators and maintaining good expansion properties in the underlying graph structure.

#### 1.44 Remark 21: GrossCodeGaugingMeasurement

When implementing quantum error correction protocols, it becomes crucial to understand the concrete overhead required for specific measurements in stabilizer codes. For the Gross code—a quantum LDPC code with favorable distance properties—measuring the logical operator  $\bar{X}_\alpha = X(\alpha f, 0)$  requires a systematic gauging construction that carefully balances the competing demands of error tolerance and resource efficiency.

The gauging procedure introduces auxiliary qubits and checks to enable transversal measurement while preserving the code distance. This construction must specify not only the structure of the auxiliary graph but also provide explicit bounds on vertex degrees and cycle lengths to ensure the resulting deformed code maintains good LDPC properties.

*Remark* (Remark 21: Gross Code Gauging Measurement). The explicit construction of the gauging graph  $G$  for measuring  $\bar{X}_\alpha = X(\alpha f, 0)$  in the Gross code yields a graph with 12 vertices (corresponding to the support of  $f$ ), 22 edges (18 matching edges plus 4 expansion edges), and 7 independent cycles. This results in a total overhead of 41 additional checks and qubits while maintaining degree bounds suitable for LDPC codes.

The construction decomposes as follows:

1. **Vertices:** The 12 monomials  $\gamma \in \mathbb{Z}_{12} \times \mathbb{Z}_6$  where  $f(\gamma) \neq 0$
2. **Matching edges:** 18 edges connecting vertices that share a  $Z$  check, determined by differences of the form  $-B_i + B_j$  where  $B = y^3 + x^2 + x$
3. **Expansion edges:** 4 carefully chosen edges  $\{((2, 0), (5, 3)), ((2, 0), (6, 0)), ((5, 3), (11, 3)), ((7, 3), (11, 3))\}$  that ensure the deformed code maintains distance 12
4. **Cycles:** 7 independent cycles of length 3 or 4, providing the flux constraints necessary for gauge invariance

The total resource overhead consists of 12 additional Gauss’s law checks  $A_v$  (one per vertex), 7 additional flux checks  $B_p$  (one per independent cycle), and 22 additional edge qubits, yielding  $12 + 7 + 22 = 41$  total additional resources.

Crucially, the maximum vertex degree is at most 6 and all cycles have length at most 4, ensuring that the Tanner graph of the deformed code retains the sparse structure essential for efficient LDPC decoding algorithms.

This construction demonstrates that logical measurements in quantum LDPC codes can be implemented with moderate overhead while preserving the code’s fundamental structural properties. The explicit nature of this result provides a concrete foundation for analyzing the practical feasibility of fault-tolerant quantum computation using LDPC codes, where the balance between measurement capability and resource efficiency is paramount.

#### 1.45 Remark 22: DoubleGrossCodeDefinition

The Double Gross code represents one of the most successful constructions of quantum LDPC codes with good parameters. Originally discovered by Gross, it achieves a remarkable balance between distance and rate while maintaining relatively low-weight stabilizer generators. This code demonstrates the power of the bivariate bicycle (BB) construction, where algebraic structure in polynomial rings translates to geometric structure in the resulting quantum code.

The gauging measurement construction provides a concrete method for measuring the logical X operators of this code, revealing the intricate relationship between the code’s algebraic description and its physical implementation requirements.

*Remark* (Remark 22: Double Gross Code Definition). The **Double Gross code** is a  $[[288, 12, 18]]$  bivariate bicycle code with parameters  $\ell = 12$ ,  $m = 12$ , defined by polynomials  $A = x^3 + y^7 + y^2$  and  $B = y^3 + x^2 + x$  in  $\mathbb{F}_2[x, y]/(x^{12} - 1, y^{12} - 1)$ .

The logical X operators take the form  $\bar{X}_\alpha = X(\alpha f, 0)$  where  $f$  is a weight-18 polynomial in the kernel of the transpose convolution operator  $B^T$ . The gauging measurement construction for these logical operators involves:

- 18 vertices (the support of  $f$ )
- 27 matching edges plus 7 expansion edges (34 total counting multiplicity)
- 13 independent flux checks (from a cycle rank of 17)

This yields a total measurement overhead of  $18 + 13 + 34 = 65$  additional resources.

The algebraic foundation of the Double Gross code rests on the polynomial ring  $\mathbb{F}_2[x, y]/(x^{12} - 1, y^{12} - 1)$ , which corresponds to the group algebra of  $\mathbb{Z}_{12} \times \mathbb{Z}_{12}$ . The choice of polynomials  $A$  and  $B$  ensures that the resulting X-type and Z-type stabilizers commute, a crucial requirement for any stabilizer code.

The polynomial  $f$  defining the logical operators satisfies  $f \in \ker(B^T)$ , meaning that  $\sum_\gamma f(\gamma) B^T(\gamma - \beta) = 0$  for all  $\beta$ . This kernel condition is precisely what ensures that the logical operators  $\bar{X}_\alpha = X(\alpha f, 0)$  commute with all Z-type stabilizers of the code.

The gauging construction transforms the measurement of logical operators into a classical post-processing problem on an auxiliary graph. The 18 vertices of this graph correspond to qubits where the logical operator acts nontrivially, while the edges encode correlations that must be tracked to maintain the logical information during measurement.

Remarkably, despite the logical operators having weight 18, the Tanner graph expansion properties are insufficient for standard distance arguments. Each qubit participates in only 3 X-type checks (corresponding to  $|\text{supp}(A)| = 3$ ), so a weight-18 error triggers at most 54 syndrome violations. This highlights why gauging provides a more robust approach to logical measurements than relying solely on syndrome extraction.

The overhead calculation reveals the true cost of fault-tolerant logical measurements: while the original code uses 288 qubits to encode 12 logical qubits, measuring a single logical operator requires 65 additional resources. This 22

## 1.46 Remark 23: Generalizations Beyond Pauli

The quantum error correction framework developed for Pauli operators on qubits extends naturally beyond this specific setting. Understanding these generalizations reveals the fundamental mathematical structures underlying gauge measurement and provides insight into when local measurements can determine global quantum charges.

*Remark* (Remark 23: Generalizations Beyond Pauli). The gauging measurement procedure generalizes beyond Pauli logical operators on qubits in three fundamental directions: (1) qudit systems where  $\mathbb{Z}_2$  is replaced by  $\mathbb{Z}_p$  for prime  $p$ , (2) abelian group charges where local measurements uniquely determine global charge, and (3) nonabelian groups where local measurements leave the global charge fundamentally ambiguous.

The first generalization replaces binary quantum systems (qubits) with  $p$ -level quantum systems (qudits). This requires extending the boundary and coboundary maps from  $\mathbb{Z}_2$  to  $\mathbb{Z}_p$  while preserving their essential linear algebraic properties.

For qudits, the boundary map  $\partial : \mathbb{Z}_p^E \rightarrow \mathbb{Z}_p^V$  computes the charge at each vertex by summing incident edge charges modulo  $p$ :

$$(\partial\gamma)_v = \sum_{\substack{e \in E \\ v \in e}} \gamma_e \pmod{p}.$$

The coboundary map  $\delta : \mathbb{Z}_p^V \rightarrow \mathbb{Z}_p^E$  operates on vertex functions, assigning to each edge the sum of its endpoint values:

$$(\delta f)_e = f(a) + f(b) \pmod{p}$$

for edge  $e = \{a, b\}$ . These maps are  $\mathbb{Z}_p$ -linear and satisfy the crucial transpose relationship: for all  $f \in \mathbb{Z}_p^V$  and  $\gamma \in \mathbb{Z}_p^E$ ,

$$\sum_{e \in E} (\delta f)_e \cdot \gamma_e = \sum_{v \in V} f_v \cdot (\partial\gamma)_v.$$

The chain complex structure generalizes as well. The second boundary map  $\partial_2 : \mathbb{Z}_p^C \rightarrow \mathbb{Z}_p^E$  assigns to each edge the sum of cycle charges from cycles containing that edge. The fundamental chain complex property  $\partial \circ \partial_2 = 0$  continues to hold, provided each cycle satisfies the incidence condition that for every vertex  $v$ , the prime  $p$  divides the number of edges in the cycle incident to  $v$ . For  $p = 2$ , this reduces to the standard requirement that cycles have even incidence at each vertex.

The second generalization concerns abelian versus nonabelian charge groups. In abelian groups, the sum of local charges is independent of measurement order due to commutativity. For any charges  $(q_1, \dots, q_n)$  and permutation  $\pi$ , we have

$$\sum_{i=1}^n q_{\pi(i)} = \sum_{i=1}^n q_i.$$

This order-independence is the mathematical foundation that makes local Gauss's law measurements work: measuring individual  $A_v$  operators in any sequence yields the same total sign  $\sigma = \sum_v \varepsilon_v$ .

The third generalization reveals a fundamental limitation when charges take values in nonabelian groups. In such groups, the product of elements depends critically on multiplication order. If  $g_1 \cdot g_2 \neq g_2 \cdot g_1$ , then measuring the local charges in different orders produces different global charges:

$$\prod [g_1, g_2] = g_1 \cdot g_2 \neq g_2 \cdot g_1 = \prod [g_2, g_1].$$

This order-dependence means that local measurements of nonabelian charges cannot uniquely determine the global charge—the measurement protocol itself affects the outcome.

These three generalizations illuminate the special role of Pauli operators on qubits: they correspond to the intersection of all three cases where the mathematics works optimally. The  $\mathbb{Z}_2$  structure provides the simplest nontrivial finite field, the abelian nature ensures measurement order independence, and the specific geometric constraints ensure the chain complex property holds. Understanding these generalizations clarifies both the power and limitations of gauge measurement procedures in quantum error correction.

### 1.47 Remark 24: ShorStyleMeasurement

The measurement procedure introduced by Shor for quantum error correction can be viewed as a special case of the general gauging framework developed in previous chapters. This perspective reveals that Shor’s approach, which uses auxiliary GHZ states and transversal controlled-X gates, naturally emerges when the gauging graph is constructed with a specific bipartite structure. The key insight is that the auxiliary qubits in Shor’s method correspond to ”dummy vertices” in the gauging graph, and the choice of how these dummy vertices are interconnected provides flexibility for hardware optimization without affecting the logical measurement outcome.

Given a Pauli logical operator  $L$  with support weight  $W = |\text{supp}(L)|$ , Shor’s measurement procedure can be realized through a gauging graph with  $2W$  vertices:  $W$  vertices corresponding to the support qubits of  $L$ , and  $W$  additional dummy vertices. Each support qubit is connected to exactly one dummy vertex (forming  $W$  ”pairing edges”), while the dummy vertices are interconnected according to an arbitrary connected subgraph  $G_{\text{dummy}}$ . This construction provides the flexibility to optimize the dummy subgraph for specific hardware constraints while preserving the measurement functionality.

*Remark* (Remark 24: Shor-Style Measurement as Gauging). The gauging measurement procedure recovers Shor-style logical measurement as a special case. Given a Pauli logical operator  $L$  with weight  $W = |\text{supp}(L)|$ , the Shor-style measurement uses an auxiliary GHZ state on  $W$  qubits, transversal CX gates, and  $X$  measurements on the auxiliary qubits. From the gauging viewpoint, one chooses a graph  $G$  with vertex set  $\text{supp}(L) \cup \{d_1, \dots, d_W\}$  where: (1)  $W$  dummy vertices  $d_i$  (one per qubit in  $\text{supp}(L)$ ); (2) a connected subgraph on the dummy vertices; (3) each  $d_i$  connected by a single edge to the corresponding qubit  $i$ . After measuring the edges of the dummy subgraph first (the ungauging step), the remaining state on dummy qubits is equivalent to a GHZ state entangled with  $\text{supp}(L)$ , recovering Shor-style measurement. The key advantage of the gauging viewpoint is that the dummy subgraph can be chosen freely (path graph, star graph, expander, etc.) to optimize for hardware constraints.

The mathematical foundation of this correspondence rests on several key structural properties. The Shor gauging graph has exactly  $2W$  vertices, with each support vertex having degree exactly 1 (connected only to its paired dummy vertex), while each dummy vertex has degree  $1 + \deg_{G_{\text{dummy}}}(i)$ , where the additional degree comes from connections within the dummy subgraph. The total number of edges is  $W + |E(G_{\text{dummy}})|$ , consisting of the  $W$  pairing edges plus the edges within the dummy subgraph.

A crucial property is that all Gauss law operators  $A_v$  on this graph are pure  $X$ -type operators and mutually commute. Their product equals the extended logical operator  $L' = \prod_{v \in V \cup D} X_v$ , which factorizes as  $L' = L \cdot \prod_{d \in D} X_d$ , where  $L$  acts on support qubits and  $\prod_d X_d$  acts on dummy qubits. When the entangling circuit transforms these Gauss operators to local Pauli  $X$  measurements, and all dummy measurements yield outcome  $+1$ , the measurement sign reduces to  $\sum_i \varepsilon_{\text{inl}(i)}$ , recovering the original logical measurement on the support qubits.

The flexibility in choosing  $G_{\text{dummy}}$  provides significant practical advantages. For hardware with limited connectivity, one might choose  $G_{\text{dummy}}$  to be a path graph (yielding a linear arrangement of dummy qubits) or a star graph (concentrating connectivity at a single central dummy vertex). For fault-tolerant implementations requiring low-weight stabilizer generators, expander graphs with bounded degree can be employed. This flexibility is absent in the original Shor formulation, which implicitly assumes a complete graph structure for the auxiliary GHZ state preparation.

### 1.48 Remark 25: SteaneStyleMeasurement

In quantum error correction, the Steane-style measurement procedure for stabilizer codes represents a fundamental technique for fault-tolerant syndrome extraction. This procedure involves preparing ancilla qubits, entangling them with data qubits, and then measuring the ancillae to extract error syndrome information. While this process may appear straightforward, its underlying mathematical structure reveals a deep connection to gauge theories on graphs and hypergraphs.

The key insight of this remark is that the entire Steane measurement can be decomposed into a sequence of three gauging operations: hypergraph gauging for state preparation, graph gauging for entanglement creation, and ungauging for measurement readout. This perspective not only provides theoretical clarity but also opens pathways for optimization and generalization of quantum error correction protocols.

*Remark* (Remark 25: Steane-Style Measurement as Gauging). For a CSS (Calderbank-Shor-Steane) stabilizer code used as an ancilla code, the complete Steane-style measurement procedure can be decomposed into three sequential gauging operations:

**Step 1 (State Preparation):** Hypergraph gauging on the Z-type check incidence relation prepares the appropriate ancilla state.

**Step 2 (Entangling):** Graph gauging on a perfect matching between data and ancilla qubits creates the necessary entanglement for syndrome extraction.

**Step 3 (Readout):** Z measurements on edge qubits (ungauging) extract the desired syndrome information.

Each step involves gauge operators that mutually commute and satisfy the algebraic requirements for their respective gauging procedures.

This decomposition demonstrates that Steane measurements arise naturally from the gauge theory framework applied to quantum error correction. The CSS structure of the ancilla code ensures that all three steps involve only compatible gauge operations, with X-type operators in the first two steps and Z-type operators in the final step. This separation is crucial for maintaining the fault-tolerant properties of the overall protocol while revealing the deep gauge-theoretic structure underlying quantum error correction.

### 1.49 Remark 26: CohenEtAlSchemeRecovery

The study of quantum error correcting codes often requires understanding the relationship between logical operators and the stabilizer group structure. When analyzing the recoverability of quantum information, a key insight from Cohen et al. is that certain geometric properties of the logical operators can guarantee successful error correction. This analysis becomes particularly important for CSS codes, where the stabilizer generators split into X-type and Z-type checks.

*Remark* (Remark 26: Cohen et al. Scheme Recovery). The Cohen et al. scheme for quantum error correction recovery exploits the hypergraph structure induced by the intersection of logical operators with Z-type stabilizer checks. For an irreducible X-type logical operator, the restricted incidence hypergraph between the logical support and relevant Z-checks has a trivial boundary kernel, enabling the construction of measurement schemes that can distinguish logical operators through their geometric properties.

The fundamental building blocks for this analysis begin with the logical support of Pauli operators. For any Pauli operator  $L$  on qubits  $Q$ , we define the **logical support** as  $\text{logicalSupport}(L) := \text{supp}_X(L)$ , which captures the qubits where  $L$  acts non-trivially with X operators.

Within a stabilizer code  $C$ , we distinguish between different types of check operators. A check index  $i$  is called a **Z-type check** if the corresponding stabilizer generator  $C.\text{check}(i)$  has no X-support, meaning its X-vector is identically zero. For a given logical operator  $L$ , the **relevant Z-checks** are those Z-type checks whose Z-support intersects the logical support:

$$E_L := \{i \in C.I \mid \text{isZTypeCheck}(i) \wedge \text{supp}_Z(C.\text{check}(i)) \cap \text{logicalSupport}(L) \neq \emptyset\}.$$

This leads to a restricted hypergraph structure where qubits in  $\text{logicalSupport}(L)$  serve as vertices and relevant Z-checks serve as hyperedges. The **restricted incidence relation** connects qubit  $q$  to check  $i$  when  $q \in \text{logicalSupport}(L)$ , check  $i$  is Z-type, and the Z-vector of  $C.\text{check}(i)$  is nonzero at  $q$ .

A crucial property of this restricted hypergraph emerges from the commutation relations in CSS codes. For any pure X-type operator  $L$  in the centralizer of  $C$ , each Z-type check has even intersection with the logical support. This follows from the symplectic inner product condition  $\langle L, C.\text{check}(i) \rangle_{\text{symp}} = 0$ , which for CSS codes simplifies to  $\sum_q L.\text{xVec}(q) \cdot (C.\text{check}(i)).\text{zVec}(q) = 0$  over  $\mathbb{F}_2$ .

The geometric structure becomes more apparent when we consider the boundary and coboundary maps of this restricted hypergraph. The **restricted boundary map**  $\partial : \mathbb{Z}_2^{E_L} \rightarrow \mathbb{Z}_2^{V_L}$  and its transpose, the **restricted coboundary map**  $\delta : \mathbb{Z}_2^{V_L} \rightarrow \mathbb{Z}_2^{E_L}$ , capture the incidence structure algebraically.

A fundamental connection exists between the kernel of the coboundary map and the centralizer of the CSS code. If a function  $f : Q \rightarrow \mathbb{Z}_2$  lies in  $\ker(\delta)$  and is supported on  $\text{logicalSupport}(L)$ , then the corresponding X-operator  $\text{prodX}(\{q \mid f(q) \neq 0\})$  belongs to the centralizer of  $C$ . This follows because elements in the kernel of  $\delta$  correspond precisely to sets with even intersection with every Z-type check's support.

The Cohen et al. scheme's power becomes evident through the following key structural result. For irreducible X-type logical operators, if the number of relevant Z-checks does not exceed the size of the logical support ( $|E_L| \leq |V_L|$ ), then the kernel of the restricted boundary map is essentially trivial. Specifically, any nonzero element in  $\ker(\partial)$  must be the all-ones vector on  $E_L$ .

This triviality result has profound implications for quantum error correction. When the boundary kernel is trivial, the restricted hypergraph has maximal expansion properties, meaning that small sets of checks have large boundaries. This geometric expansion translates directly into the ability to distinguish between different logical operators through syndrome measurements, forming the foundation of the Cohen et al. recovery scheme.

The scheme extends naturally to layered constructions, where multiple copies of the restricted hypergraph are connected through path graph structures. These layered constructions preserve the expansion properties while enabling more sophisticated measurement protocols. The key insight is that the Gauss subset product over the layered incidence relation produces pure X-type operators that can measure arbitrary linear combinations of logical operators across different layers.

Through careful analysis of the Cheeger constant of the underlying path graphs, one can bound the expansion properties of these layered constructions, providing explicit parameters for the recovery scheme's performance. This geometric approach to quantum error correction demonstrates how topological and combinatorial methods can yield practical algorithms for quantum information processing.