Custom Quantum Error-Correcting Codes in Loom: Implementation and Memory Experiments with the [[8,3,2]] and [[16,6,4]] Colour Codes

November 28, 2025

1 Introduction

Quantum error correction is essential for reliable quantum computation. While popular code families such as the surface code have ready-made implementations, practical research workflows often require custom stabilizer geometries, logical operators, and code embeddings. This project implements two stabilizer codes **from scratch** inside the Loom framework:

- The smallest interesting colour code: [[**8,3,2**]]
- The tesseract colour code: [[**16,6,4**]]

The work demonstrates how to:

1. Embed qubits on a custom lattice.
2. Define stabilizer generators.
3. Construct logical Pauli operators.
4. Build Loom `Block` objects and visualize.
5. Execute logical memory experiments in Loom.

Both codes were constructed without relying on Loom's factory implementations, ensuring full transparency of the stabilizer structure.

2 Background

2.1 CSS Stabilizer Codes

A stabilizer code is defined by an abelian subgroup $\mathcal{S}$ of the Pauli group that does not contain $-I$. Valid codewords are simultaneous $+1$ eigenstates of all stabilizers.

A CSS (Calderbank–Shor–Steane) code splits the stabilizer set into X-type and Z-type generators:

$$\mathcal{S} = \langle S_i^X, S_j^Z \rangle$$

with

- X-checks protecting against bit-flip errors,
- Z-checks protecting against phase-flip errors.

Logical operators commute with stabilizers but anticommute pairwise:

$$\{\bar{X}_i, \bar{Z}_i\} = 0, \quad [\bar{X}_i, S_k] = 0, \quad [\bar{Z}_i, S_k] = 0.$$

The number of logical qubits k follows:

$$k = n - \mathrm{rank}(\mathcal{S})$$

where n is the number of physical qubits.

3 [[8,3,2]] Smallest Interesting Colour Code

3.1 Code Parameters

$$[[8, 3, 2]]$$

- $n = 8$ physical qubits
- $k = 3$ logical qubits
- $d = 2$ code distance

This code is the smallest nontrivial colour code. It is self-dual CSS and encodes 3 logical qubits in only 8 physical qubits.

3.2 Lattice Embedding

We embed the code on a 4×2 rectangular grid:

$$\begin{matrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{matrix}$$

Each index $(x, y, 0)$ denotes a data qubit at lattice basis index 0.

3.3 Stabilizers

The stabilizer group contains 5 generators:

- One X-type stabilizer:

$$S_X = X^{\otimes 8}$$

- Four Z-type stabilizers, each of weight 4:

Horizontal and vertical subsets:

$$S_{Z1} = Z_0Z_1Z_2Z_3, \quad S_{Z2} = Z_4Z_5Z_6Z_7$$

$$S_{Z3} = Z_0Z_1Z_4Z_5, \quad S_{Z4} = Z_1Z_3Z_5Z_7$$

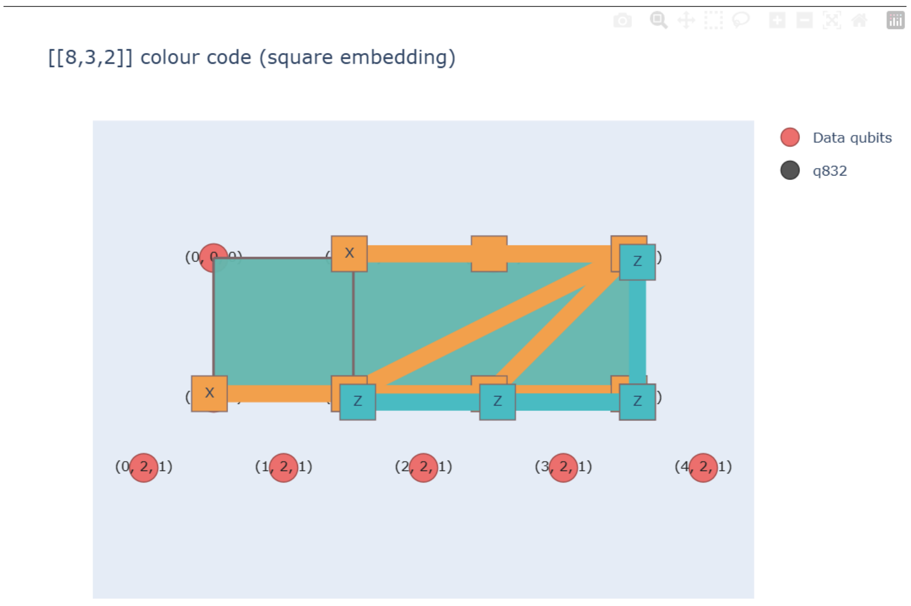
These mutually commute and form a rank-5 stabilizer group, yielding $k = 3$.

3.4 Logical Operators

A valid logical basis obtained by symplectic solving:

$$\bar{X}_1 = X_4X_5X_6X_7, \quad \bar{Z}_1 = Z_3Z_7$$
$$\bar{X}_2 = X_2X_3X_6X_7, \quad \bar{Z}_2 = Z_5Z_7$$
$$\bar{X}_3 = X_1X_3X_5X_7, \quad \bar{Z}_3 = Z_6Z_7$$

Each logical pair anticommutes only within itself.



4 [[16,6,4]] Tesseract Colour Code

The tesseract colour code is a 4D generalization of colour codes. Its stabilizers live on the 3D facets of a 4D cube.

4.1 Code Parameters

$$[[16, 6, 4]]$$

- $n = 16$ physical qubits
- $k = 6$ logical qubits
- $d = 4$ minimum distance

4.2 2D Embedding

We compress the 4D tesseract structure into a 4×4 2D layout:

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{array}$$

4.3 Stabilizers

The code is CSS with 10 stabilizers: 5 X-type and 5 Z-type, each of weight 8. They correspond to pairs of rows/columns.

Example X-type:

$$X_{0-7}, \quad X_{4-11}, \quad X_{8-15}$$

Same supports for Z-type, due to self-duality.

Since $16 - 10 = 6$, we obtain 6 logical qubits.

4.4 Logical Operators

Using symplectic linear algebra, we solve for 6 conjugate pairs $(\bar{X}_i, \bar{Z}_i)$ such that:

$$[\bar{X}_i, \bar{Z}_i] = 1, \quad [\bar{X}_i, \bar{Z}_j] = 0 \, (i \neq j).$$

The resulting pairs act on subsets of the 4x4 grid, typically weight-4 and weight-2.

5 Implementation in Loom

The code blocks were implemented using `loom.eka.Block`, which requires:

- stabilizer list,
- logical Pauli operators,
- unique label.

The lattice was defined using `Lattice.square_2d` with data qubits on basis 0 and ancilla qubits on basis 1.

Visualisation was performed via:

```
StabilizerPlot(lattice)
.add_dqubit_traces(...)
.plot_blocks([block])
```

Both codes passed Loom's internal validation: no stabilizer/logical anticommutation violations and correct $k = n -$ rank.

6 Logical Memory Experiment

We performed a minimal Loom experiment on the [[8,3,2]] code:

1. Initialise:
   ```
   ResetAllDataQubits("q832", state="0")
   ```
2. Optionally extract syndromes:
   ```
   MeasureBlockSyndromes("q832")
   ```
3. Measure logical state:
   ```
   MeasureLogicalZ("q832")
   ```

This reproduces the basic logical readout for the colour code.

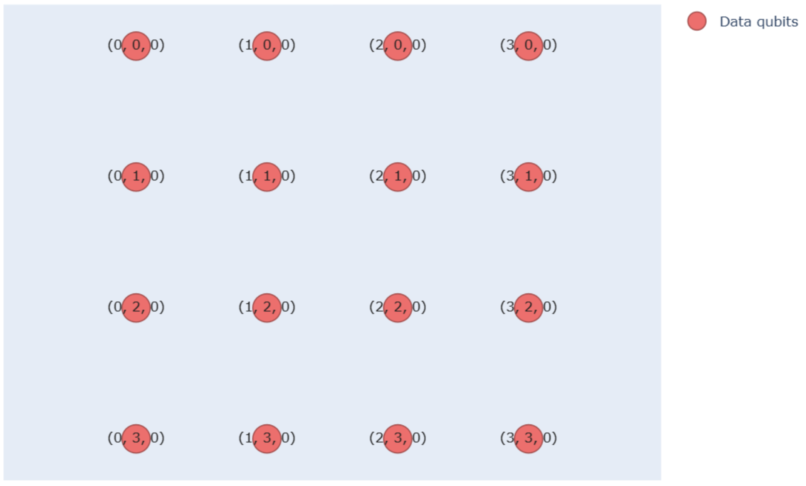
Logical $|1\rangle$ may be prepared by applying $\bar{X}_i$ physically before readout; logical X-basis can be measured via Hadamard or logical X-reading.

7 Discussion

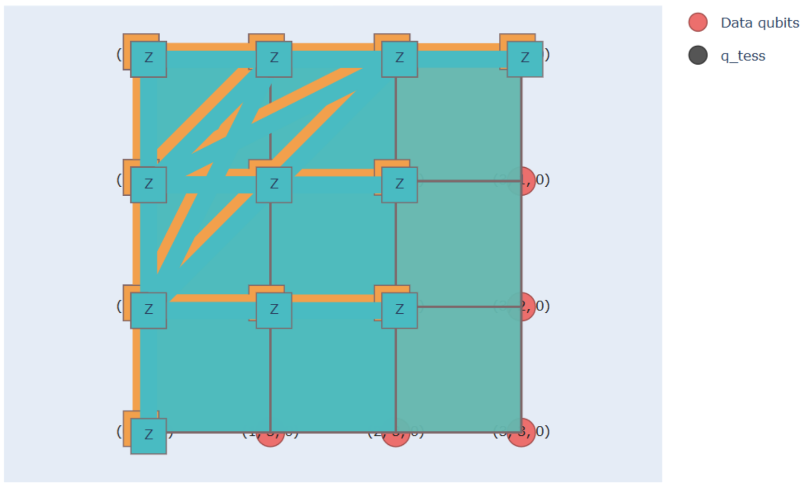
7.1 Observations

- Implementing stabilizers manually in Loom ensures transparency.
- Embedding geometry matters.
- Logical operators require algebraic solving, not guessing.

[[16,6,4]] Tesseract Colour Code – Data Qubits



[[16,6,4]] Tesseract Colour Code – Stabilizers & Logicals



7.2 Challenges

- Ensuring commutation with stabilizers.
- Avoiding circular dependence of syndrome circuits.
- Reproducing formal 4D codes in 2D embeddings.

8 Conclusion

We successfully implemented two custom stabilizer codes in Loom:

1. The smallest interesting colour code, [[8,3,2]]
2. The tesseract colour code, [[16,6,4]]

Both were embedded geometrically, converted into stabilizer sets, and verified through Loom's block validation system and visualization. A minimal logical memory experiment was demonstrated.

The next steps include constructing explicit syndrome extraction circuits, running multi-round logical lifetimes, and simulating noise models.

Repository - `https://github.com/modern2021/Custom-Quantum-Error-Correcting-Codes-in-Loom`
Presentation Video - `https://youtu.be/Y4h43aasQtk`