Quantum Error Correction Learning Tool
an Educational Tool for Visualizing Error Correction Codes

Serxhio Dosku
Quantum Computation and Error Correction, Fall Semester 2025
Department of Mathematics and Computer Science
University of Basel
Basel, Switzerland

January 8, 2026

Abstract

Quantum error correction (QEC) is a fundamental requirement for practical quantum computing, addressing the inherent fragility of quantum information. This project implements and analyzes two fundamental QEC codes: the 3-qubit bit-flip code and the 5-qubit perfect code. We develop a comprehensive visualization tool that demonstrates the complete error correction workflow, from encoding logical qubits through error detection, syndrome measurement, correction, and decoding. The implementation supports multiple error models including discrete Pauli errors (bit-flip, phase-flip, combined) and continuous rotation errors. Through quantitative fidelity analysis, we validate the theoretical predictions and demonstrate the capabilities and limitations of each code. Our tool provides an accessible educational platform for understanding QEC principles while maintaining scientific rigor in implementation and analysis.

Contents

1 Introduction

1.1 Problem Statement and Motivation

Quantum computers promise revolutionary computational capabilities, from factoring large numbers to simulating complex quantum systems. However, quantum information is extraordinarily fragile. Unlike classical bits that exist in well-defined 0 or 1 states, quantum bits (qubits) exist in superpositions $\alpha|0\rangle + \beta|1\rangle$, making them extremely susceptible to environmental noise, decoherence, and operational errors.

The challenge of quantum error correction differs fundamentally from classical error correction in several critical ways:

1. **No-cloning theorem:** Quantum information cannot be copied, precluding the simple redundancy schemes that work for classical systems. This fundamental constraint, means we cannot simply duplicate qubits to protect information.
2. **Continuous error space:** While classical errors are discrete (bit-flips), quantum errors can be continuous rotations or general unitary transformations, creating an infinite-dimensional error space.
3. **Measurement disturbance:** Direct measurement of quantum states collapses superpositions, destroying the very information we seek to protect. Error detection must therefore use indirect methods through ancilla qubits and stabilizer measurements.
4. **Quantum gate errors:** Unlike classical gates which are typically deterministic, quantum gates are subject to noise, making even error correction operations themselves prone to errors (necessitating fault-tolerant protocols).

These constraints necessitate sophisticated error correction codes that encode logical qubits into multiple physical qubits while preserving quantum information through carefully designed stabilizer operators. Without effective error correction, practical quantum computing remains impossible.

1.2 Project Goals and Scope

This project aims to bridge the gap between theoretical understanding and practical implementation of quantum error correction. Our primary objectives are:

1. **Implementation:** Develop complete, functional implementations of two fundamental QEC codes—the 3-qubit bit-flip code and the 5-qubit perfect code—with full encoding, syndrome measurement, correction, and decoding capabilities.
2. **Visualization:** Create an interactive visualization tool that demonstrates the complete QEC workflow, making abstract quantum concepts tangible through visual representation.
3. **Analysis:** Perform quantitative analysis of error correction effectiveness using fidelity metrics, validating theoretical predictions and exploring practical limitations.
4. **Education:** Provide an accessible educational platform that enables both beginners and advanced users to explore QEC principles through hands-on experimentation.

Our implementation focuses on single-qubit error correction, which forms the foundation for more complex error correction schemes. The 3-qubit bit-flip code serves as the simplest example, while the 5-qubit perfect code demonstrates the smallest code capable of correcting arbitrary single-qubit errors. This scope is appropriate for a course project, providing comprehensive understanding of QEC fundamentals while remaining computationally tractable.

1.3 Report Structure

This report follows a standard research paper structure: Section 2 presents the theoretical background necessary to understand quantum error correction; Section 3 describes our implementation methodology and architecture; Section 4 presents results and quantitative analysis; Section 5 discusses limitations and future directions; and Section 6 provides concluding remarks.

2 Theoretical Background

2.1 Quantum States and the Need for Error Correction

A single qubit state is described by a complex superposition:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

where $\alpha, \beta \in \mathbb{C}$ satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. The state exists in a two-dimensional Hilbert space, and any unitary operation or measurement can alter this state. Environmental interactions, imperfect gate operations, and decoherence introduce errors that corrupt quantum information.

Quantum errors can be classified into discrete Pauli errors, which form the foundation of stabilizer codes. The three Pauli operators are:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{(Bit-flip)} \tag{2}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \text{(Phase-flip)} \tag{3}$$

$$Y = iXZ = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \text{(Combined)} \tag{4}$$

These operators form a basis for single-qubit errors: any single-qubit error can be expressed as a linear combination of X, Y, and Z operators. The X operator flips the computational basis ($|0\rangle \leftrightarrow |1\rangle$), the Z operator flips the phase ($|+\rangle \leftrightarrow |-\rangle$ where $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$), and Y combines both effects.

2.2 Stabilizer Formalism

Stabilizer codes, first developed by Gottesman, provide a powerful framework for quantum error correction. A stabilizer S is an operator that leaves encoded logical states unchanged:

$$S|\psi_L\rangle = |\psi_L\rangle \tag{5}$$

For an $[n, k]$ quantum error correction code that encodes k logical qubits into n physical qubits, we require $n - k$ independent stabilizer generators. The stabilizers generate an Abelian subgroup of the Pauli group, and the code space is the simultaneous $+1$ eigenspace of all stabilizers.

Errors that anti-commute with stabilizers can be detected through syndrome measurement. The syndrome is a classical bitstring obtained by measuring each stabilizer, indicating which stabilizers were violated. This syndrome uniquely identifies the error (up to equivalence classes of errors that have the same effect on the code space).

2.3 3-Qubit Bit-Flip Code

The 3-qubit bit-flip code, introduced by Shor, is the simplest quantum error correction code. It encodes one logical qubit into three physical qubits:

$$|0_L\rangle = |000\rangle \tag{6}$$
$$|1_L\rangle = |111\rangle \tag{7}$$

The code uses two stabilizer generators: Z_0Z_1 and Z_1Z_2, which measure the parity between adjacent qubit pairs. These stabilizers detect bit-flip (X) errors because X errors anti-commute with Z measurements.

The syndrome measurement yields a 2-bit syndrome:

- $[0, 0]$: No error detected
- $[1, 0]$: Error on qubit 0 (first parity violated)
- $[0, 1]$: Error on qubit 2 (second parity violated)
- $[1, 1]$: Error on qubit 1 (both parities violated)

The encoding circuit simply copies the logical qubit: if the logical state is $|1\rangle$, apply X to qubit 0, then CNOT gates from qubit 0 to qubits 1 and 2. This creates the repetition code structure.

Limitations: This code only corrects X errors. Phase-flip (Z) errors commute with the Z-stabilizers and are therefore undetectable. The code has distance 3 (can detect 2 errors, correct 1), but only for the specific error type it's designed for.

2.4 5-Qubit Perfect Code

The 5-qubit perfect code, discovered independently by Laflamme et al. [?] and Bennett et al. [?], is remarkable: it is the smallest quantum code capable of correcting arbitrary single-qubit errors. It achieves the theoretical lower bound given by the quantum Hamming bound.

The code encodes one logical qubit into five physical qubits and uses four stabilizer generators, yielding a 4-bit syndrome that uniquely identifies any single-qubit Pauli error (15 non-trivial error possibilities plus the no-error case = 16 total syndromes).

The stabilizer generators for the 5-qubit code are:

$$S_1 = X_1Z_2Z_3X_4 \tag{8}$$
$$S_2 = X_2Z_3Z_4X_0 \tag{9}$$
$$S_3 = X_3Z_4Z_0X_1 \tag{10}$$
$$S_4 = X_4Z_0Z_1X_2 \tag{11}$$

Each stabilizer is a product of X and Z operators, enabling detection of all Pauli error types. The encoding circuit involves Hadamard gates and controlled-Z operations to create the necessary entanglement structure.

This code represents a significant improvement over the bit-flip code, as it corrects *any* single-qubit Pauli error (X, Y, or Z) on any of the five qubits, making it truly universal for single-error correction.

2.5 Fidelity as a Performance Metric

Fidelity provides a quantitative measure of similarity between quantum states. For two density matrices ρ and σ, the fidelity is defined as:

$$F(\rho,\sigma) = \left(\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2 \tag{12}$$

For pure states $|\psi\rangle$ and $|\phi\rangle$, this simplifies to:

$$F = |\langle\psi|\phi\rangle|^2 \tag{13}$$

Fidelity ranges from 0 (orthogonal states) to 1 (identical states). In our implementation, we compute fidelity between the ideal encoded state and the state after error injection (before correction) and after correction (after decoding). Perfect error correction should restore fidelity to 1.0, while imperfect correction or uncorrectable errors result in lower fidelity values.

3 Methods and Implementation

3.1 System Architecture

Our implementation follows a modular architecture with clear separation of concerns, enabling extensibility and maintainability. The system consists of four main components:

1. **QEC Codes Module** (`qec_codes.py`): Implements the abstract `QECCode` base class and concrete implementations for `BitFlipCode` and `PerfectCode`. Each code provides methods for encoding, syndrome measurement, correction, and decoding.
2. **Error Injection Module** (`error_injection.py`): Handles injection of various error types (bit-flip, phase-flip, depolarizing, rotation errors) into quantum circuits. Supports probabilistic error injection and custom parameters.
3. **Backend Module** (`backend.py`): Performs state vector calculations using Qiskit's quantum simulation, extracts syndromes from measurement results, computes fidelity between states, and orchestrates the complete QEC workflow.
4. **Visualization Interface** (`frontend.py`, `visualizer.py`): Provides interactive web-based interface using Streamlit, along with programmatic visualization tools for circuits, state vectors, and 3D spatial representations.

3.2 Code Implementation Details

3.2.1 3-Qubit Bit-Flip Code Implementation

The encoding circuit for the bit-flip code follows the straightforward repetition code structure. Given a logical state (0 or 1), we prepare the first qubit in that state, then use CNOT gates to copy it to the other two qubits:

Listing 1: 3-Qubit Bit-Flip Code Encoding

```python
def encode(self, logical_state: int = 0) -> QuantumCircuit:
    qreg = QuantumRegister(3, 'q')
    creg = ClassicalRegister(3, 'c')
    circuit = QuantumCircuit(qreg, creg)

    # Prepare logical state
    if logical_state == 1:
        circuit.x(qreg[0])

    # Encoding: |0   -> |000   , |1   -> |111
    circuit.cx(qreg[0], qreg[1])
    circuit.cx(qreg[0], qreg[2])

    return circuit
```

Syndrome measurement uses parity measurements. For stabilizer Z_0Z_1, we apply CNOT gates to copy parity information to an ancilla, measure, then uncompute. The implementation from `qec_codes.py`:

Listing 2: Syndrome Measurement for 3-Qubit Bit-Flip Code

```
def syndrome_measurement(self, qreg, creg):
    circuit = QuantumCircuit(qreg, creg)

    # Measure Z_0 Z_1 (parity of qubits 0 and 1)
    circuit.cx(qreg[0], qreg[1])
    circuit.measure(qreg[1], creg[0])
    circuit.cx(qreg[0], qreg[1])  # Uncompute to restore q1

    # Measure Z_1 Z_2 (parity of qubits 1 and 2)
    circuit.cx(qreg[1], qreg[2])
    circuit.measure(qreg[2], creg[1])
    circuit.cx(qreg[1], qreg[2])  # Uncompute to restore q2

    return circuit
```

Correction is straightforward: based on the 2-bit syndrome, we apply an X gate to the appropriate qubit. The correction logic from `qec_codes.py`:

Listing 3: Correction Logic for 3-Qubit Bit-Flip Code

```
def correct(self, syndrome: List[int]) -> QuantumCircuit:
    qreg = QuantumRegister(3, 'q')
    circuit = QuantumCircuit(qreg)

    if len(syndrome) != 2:
        raise ValueError("Syndrome␣must␣be␣a␣2-bit␣value")

    # Determine error location
    if syndrome == [1, 0]:
        circuit.x(qreg[0])  # Error on qubit 0
    elif syndrome == [0, 1]:
        circuit.x(qreg[2])  # Error on qubit 2
    elif syndrome == [1, 1]:
        circuit.x(qreg[1])  # Error on qubit 1
    # [0, 0] means no error, no correction needed

    return circuit
```

3.2.2 5-Qubit Perfect Code Implementation

The 5-qubit perfect code implementation is more complex. The encoding circuit involves a sequence of Hadamard and controlled-Z gates arranged in a specific pattern to create the required entanglement structure. The stabilizer measurement circuit applies the four stabilizer generators using ancilla qubits and controlled operations.

The correction lookup table maps 16 possible syndromes to specific correction operations. This table was constructed empirically through systematic testing: for each possible single-qubit error type and location, we measured the resulting syndrome and recorded the mapping. The correction operations involve applying the appropriate Pauli operator (X, Y, or Z) to the affected qubit.

3.3 Error Injection System

Our error injection system supports multiple error models:

- **Discrete Pauli Errors:** X (bit-flip), Z (phase-flip), Y (combined), and depolarizing (probabilistic mixture).
- **Continuous Rotation Errors:** $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$ gates with custom rotation angles. These represent realistic decoherence processes.
- **Error Parameters:** Supports probabilistic error injection, multiple simultaneous errors, and custom qubit selection.

The `ErrorInjector` class handles error injection by composing error gates into the quantum circuit at the specified locations. Core implementation from `error_injection.py`:

Listing 4: Error Injection Implementation

```python
def inject_error(self, circuit: QuantumCircuit, error_type: ErrorType,
                 qubit: int, error_probability: float = 1.0,
                 rotation_angle: float = None) -> QuantumCircuit:
    error_circuit = circuit.copy()

    # Apply error with given probability
    if error_probability >= 1.0 - 1e-10:
        should_apply = True
    else:
        should_apply = np.random.random() < error_probability

    if should_apply:
        if error_circuit.qregs:
            qreg = error_circuit.qregs[0]
            target_qubit = qreg[qubit]

        if error_type == ErrorType.BIT_FLIP:
            error_circuit.x(target_qubit)
        elif error_type == ErrorType.PHASE_FLIP:
            error_circuit.z(target_qubit)
        elif error_type == ErrorType.Y_ERROR:
            error_circuit.y(target_qubit)
        elif error_type == ErrorType.ROTATION_X:
            angle = rotation_angle if rotation_angle is not None else np.pi / 4
            error_circuit.rx(angle, target_qubit)
        elif error_type == ErrorType.ROTATION_Y:
            angle = rotation_angle if rotation_angle is not None else np.pi / 4
            error_circuit.ry(angle, target_qubit)
        elif error_type == ErrorType.ROTATION_Z:
            angle = rotation_angle if rotation_angle is not None else np.pi / 4
            error_circuit.rz(angle, target_qubit)

    return error_circuit
```

3.4 Syndrome Extraction and Correction

Syndrome extraction involves composing the error circuit with the syndrome measurement circuit and running the simulation. The measurement results yield a classical bitstring (syndrome) that identifies the error. Our backend extracts this syndrome using the implementation from `backend.py`:

Listing 5: Syndrome Extraction from Measurement Results

```
def extract_syndrome(self, circuit: QuantumCircuit,
                     syndrome_measurement_circuit: QuantumCircuit) -> List[int]:
    qreg = circuit.qregs[0]
    full_circuit = QuantumCircuit(qreg)

    # Copy all quantum operations (skip measurements)
    for instruction in circuit.data:
        if instruction.operation.name != 'measure':
            full_circuit.append(instruction.operation, instruction.qubits)

    # Create classical register for syndrome
    n_syndromes = syndrome_measurement_circuit.num_clbits
    creg = ClassicalRegister(n_syndromes, 'syndrome')
    full_circuit.add_register(creg)

    # Rebuild syndrome measurement using the same quantum register
    for instruction in syndrome_measurement_circuit.data:
        op = instruction.operation
        qubits = instruction.qubits
        clbits = instruction.clbits
        new_qubits = [qreg[q._index] for q in qubits if q._index < len(qreg)]
        new_clbits = [creg[c._index] for c in clbits if c._index < n_syndromes]
        if new_qubits and new_clbits:
            full_circuit.append(op, new_qubits, new_clbits)

    # Run simulation with multiple shots
    job = self.simulator.run(full_circuit, shots=1024)
    result = job.result()
    counts = result.get_counts()

    # Extract most probable syndrome
    if counts:
        bitstring = max(counts, key=counts.get)
        bitstring_clean = ''.join(c for c in bitstring if c in '01')
        if len(bitstring_clean) >= n_syndromes:
            syndrome = [int(bit) for bit in reversed(bitstring_clean[:
                n_syndromes])]
            return syndrome

    return [0] * n_syndromes
```

To quantify error correction effectiveness, we compute fidelity between ideal and corrupted states using the implementation from `backend.py`:

Listing 6: Fidelity Calculation Between Quantum States

```
def calculate_fidelity(self, ideal_state: Statevector,
                       noisy_state: Statevector) -> float:
    """
␣␣␣␣Calculate␣fidelity␣between␣ideal␣and␣noisy␣states.
␣␣␣␣Fidelity␣=␣|    _ideal    |    _noisy    |
␣␣␣␣"""
    # Calculate overlap
    overlap = ideal_state.inner(noisy_state)
    fidelity = abs(overlap) ** 2
    return fidelity
```

Correction is applied based on the syndrome using a lookup table. For the 3-qubit code, this is a simple 4-entry table. For the 5-qubit code, we use a 16-entry table covering all possible single-qubit errors.

3.5 Visualization and User Interface

We provide multiple interfaces for different user needs:

- **Streamlit Web Interface:** Interactive step-by-step workflow with real-time visualizations, parameter selection widgets, and educational explanations.
- **Jupyter Notebook:** Interactive widgets for experimentation and learning.
- **Programmatic API:** Python scripts for batch processing and automated analysis.

Visualizations include:

- Circuit diagrams at each stage (encoding, error, syndrome, correction, decoding)
- State vector probability distributions
- 2D/3D spatial representations of qubits and stabilizers (PanQEC-inspired style)
- Fidelity evolution charts
- Syndrome lookup tables

3.6 Technology Stack

Our implementation uses:

- **Python 3.8+**: Core programming language
- **Qiskit 0.45.0+**: Quantum circuit construction and simulation
- **Qiskit Aer**: State vector simulation backend
- **NumPy**: Numerical computations
- **Matplotlib/Plotly**: Visualization (static and interactive)
- **Streamlit**: Web-based user interface

4 Results and Analysis

4.1 3-Qubit Bit-Flip Code Performance

We tested the 3-qubit bit-flip code with various error scenarios. For a bit-flip error on qubit 0 with logical state $|1_L\rangle = |111\rangle$, the results demonstrate perfect error correction:

Table 1: Fidelity Analysis for 3-Qubit Bit-Flip Code (Bit-Flip Error)

Stage	Fidelity
After Encoding	1.0000
After Error (X on q0)	0.0000
After Correction	1.0000

The code successfully detects and corrects the error, restoring fidelity to 1.0. However, as expected from theory, the code fails for phase-flip errors: a Z error on any qubit results in fidelity 0.854 (for rotation angle $\pi/4$) even after correction, since the stabilizers cannot detect phase errors.

4.2 5-Qubit Perfect Code Performance

The 5-qubit perfect code demonstrates correction capability for all single-qubit Pauli errors. Table 2 shows results for different error types:

Table 2: Error Correction Performance for 5-Qubit Perfect Code

Error Type	Fidelity (After Error)	Fidelity (After Correction)
Bit-flip (X)	0.0000	1.0000
Phase-flip (Z)	0.0000	1.0000
Y error	0.0000	1.0000

The perfect code successfully corrects all discrete single-qubit Pauli errors, validating its theoretical properties. The code correctly identifies errors through syndrome measurement and applies appropriate corrections, restoring the encoded state with perfect fidelity.

4.3 Rotation Error Analysis

For continuous rotation errors, the results differ significantly. Rotation errors create continuous deviations from the code space, which discrete stabilizer codes cannot perfectly correct. Table 3 shows results for $R_x(\pi/4)$ rotation errors:

Table 3: Rotation Error Correction (R_x gate, $\theta = \pi/4$)

Code	Fidelity (After Error)	Fidelity (After Correction)
3-Qubit Bit-Flip	0.854	0.854
5-Qubit Perfect	0.854	0.854

Both codes show identical behavior: the fidelity after correction equals the fidelity after error, indicating that the correction procedure does not improve the state. This is theoretically expected: stabilizer codes are designed for discrete Pauli errors, not continuous rotations. Small rotation angles can be approximated as discrete errors, but perfect correction is impossible.

This highlights a fundamental limitation: realistic quantum systems experience continuous decoherence, but discrete codes provide only approximate correction. This motivates the need for fault-tolerant protocols and error suppression techniques in addition to error correction.

4.4 Syndrome Analysis

Our syndrome extraction correctly identifies error locations. For the 3-qubit code, we validated the syndrome lookup table through systematic testing. Table 4 shows the complete mapping:

Table 4: Syndrome Lookup Table for 3-Qubit Bit-Flip Code

Syndrome	Correction
00	No error
10	X on qubit 0
01	X on qubit 2
11	X on qubit 1

For the 5-qubit code, all 16 syndromes are correctly mapped to appropriate corrections, covering all 15 non-trivial single-qubit errors plus the no-error case.

4.5 Fidelity Evolution

Throughout the QEC process, fidelity evolves as expected:

1. **After Encoding:** Fidelity = 1.0 (perfect encoded state)
2. **After Error Injection:** Fidelity drops (error corrupts the state)
3. **After Correction:** Fidelity recovers (for correctable errors) or remains low (for uncorrectable errors)

This evolution provides quantitative validation of the error correction mechanism and enables comparison between different codes and error types.

4.6 Code Validation

We developed a comprehensive test suite (`tests/test_perfect_code_errors.py`) covering 14 test cases for the 5-qubit perfect code, testing all error types (bit-flip, phase-flip, Y errors, depolarizing, and rotation errors for R_x, R_y, R_z). All tests validate that:

- Syndrome detection works correctly for all error types
- Correction lookup table is complete and accurate
- Fidelity calculations are consistent
- Discrete Pauli errors are corrected with high fidelity
- Rotation errors show expected partial correction (theoretical limitation)

5 Limitations and Discussion

5.1 Theoretical Limitations

Our implementation, while functionally correct, is subject to several theoretical limitations:

1. **Discrete vs. Continuous Errors:** Stabilizer codes correct discrete Pauli errors (X, Y, Z) but not continuous rotation errors. Real quantum systems experience continuous decoherence, creating a mismatch between error models and correction capabilities. Small rotations can be approximated, but perfect correction is impossible.
2. **Single Error Correction:** Both codes correct only single-qubit errors. Multiple simultaneous errors can cause code failure. Real quantum systems experience errors continuously, necessitating more sophisticated codes (e.g., surface codes) with error thresholds.
3. **Code Distance:** The 3-qubit code has distance 3 (detects 2 errors, corrects 1), while the 5-qubit code also has distance 3 but corrects all single-qubit error types. Larger codes with greater distance are needed for higher error rates.
4. **Measurement Errors:** Our implementation assumes perfect syndrome measurement. In practice, measurement operations themselves are noisy, requiring fault-tolerant measurement protocols.

5.2 Implementation Limitations

Our simulation-based implementation has several practical limitations:

1. **Perfect Simulation:** We use ideal quantum simulation (state vector evolution). Real hardware introduces additional noise in gates, measurements, and qubit interactions that our simulation does not capture.
2. **Fixed Error Models:** We assume known error types and locations for demonstration. Real quantum systems experience unknown, probabilistic errors with spatial and temporal correlations.
3. **No Fault Tolerance:** Our implementation doesn't account for errors in the error correction process itself. Fault-tolerant QEC requires additional layers of encoding and carefully designed measurement protocols.
4. **Finite Precision:** Numerical simulations have finite precision, though this is negligible for our purposes.

5.3 Scaling Considerations

As quantum systems scale, overhead becomes significant:

- 3-qubit code: $3\times$ overhead for single error correction
- 5-qubit code: $5\times$ overhead for arbitrary single error correction
- Larger codes (e.g., surface codes): Require even more overhead but provide better error thresholds and can correct multiple errors

The overhead must be balanced against error rates: higher overhead enables lower error thresholds, making computation possible with noisier hardware.

5.4 Future Directions

Potential extensions and improvements include:

- **Additional Codes:** Implementation of Shor code (9-qubit), Steane code (7-qubit), or surface codes (2D/3D topological codes) for comparison.
- **Fault Tolerance:** Integration of fault-tolerant protocols that account for errors in error correction operations.
- **Real Hardware:** Integration with real quantum hardware (IBM Quantum, Google Quantum AI) to test codes under realistic noise conditions.
- **Error Threshold Analysis:** Calculation of error thresholds—the maximum error rate below which error correction improves fidelity.
- **Advanced Decoders:** Implementation of more sophisticated decoders (e.g., minimum-weight perfect matching for surface codes) that can handle multiple errors.
- **Noise Model Calibration:** Integration of realistic noise models calibrated from hardware characterization data.

6 Conclusion

This project successfully implements and analyzes two fundamental quantum error correction codes, providing both theoretical understanding and practical demonstration of QEC principles. Our interactive visualization tool effectively demonstrates the complete error correction workflow, from encoding through error detection and correction.

Key findings:

1. The 3-qubit bit-flip code successfully corrects single bit-flip errors, achieving perfect fidelity recovery, but fails for phase-flip errors as theoretically predicted.
2. The 5-qubit perfect code corrects arbitrary single-qubit Pauli errors (X, Y, Z), validating its theoretical design as the smallest universal single-error correction code.
3. Both codes fail to perfectly correct continuous rotation errors, highlighting the distinction between discrete and continuous error models and the limitations of stabilizer codes for realistic decoherence.
4. Fidelity metrics provide quantitative validation of error correction effectiveness, enabling comparison between codes and error types.

The project achieves its educational goals by providing clear visualization and analysis of QEC mechanisms, while demonstrating the practical challenges and limitations of error correction in quantum computing. The implementation serves as a foundation for understanding more complex error correction schemes used in current quantum computing research, such as surface codes and fault-tolerant protocols.

Our modular architecture and extensible design enable future enhancements, and the comprehensive documentation supports reproducibility and educational use. The tool bridges the gap between theory and practice, making complex QEC concepts accessible through interactive exploration.

7 References

1. Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press.
2. Laflamme, R., Miquel, C., Paz, J. P., & Zurek, W. H. (1996). Perfect quantum error correcting code. *Physical Review Letters*, 77(1), 198–201.
3. Shor, P. W. (1995). Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4), R2493–R2496.
4. Steane, A. M. (1996). Error correcting codes in quantum theory. *Physical Review Letters*, 77(5), 793–797.
5. Gottesman, D. (1997). *Stabilizer codes and quantum error correction* (Doctoral dissertation, California Institute of Technology).
6. Qiskit Development Team. (2024). Qiskit: An Open-source Framework for Quantum Computing. https://qiskit.org/