Tackling Noise Using Error Correction

21k-4680   
Syed Muhammad Taha Hassan21k-4678  
Asad Noor Khan

***Abstract*—Quantum error correction is a critical area of research in quantum computing, aiming to address the challenges posed by noise and imperfections in quantum systems. This project explores the implementation of repetition codes for error correction, where logical qubits are encoded across multiple physical qubits to detect and correct bit-flip errors. Using Qiskit, a quantum computing framework, the project simulates the impact of bit-flip errors and applies a majority voting technique to decode the logical qubit. The error correction is evaluated using a quantum circuit running on a simulator, allowing for an analysis of the system's performance under different noise conditions. Additionally, the project investigates advanced decoding strategies such as lookup table decoding to enhance error resilience. The findings provide insights into the effectiveness of repetition codes in improving the reliability and fault tolerance of quantum computations, offering a fundamental approach to error correction that is crucial for the future scalability of quantum computing technologies.**

Introduction

Quantum computing relies on encoding information in qubits, which are the fundamental units of quantum information. Most quantum algorithms developed over the past few decades have assumed that these qubits are ideal, meaning they can be prepared in any desired state and manipulated with complete precision. These idealized qubits are referred to as *logical qubits*. However, in practice, the physical qubits used in real quantum systems are subject to imperfections such as decoherence and gate errors. Despite continuous advances in developing better quality physical qubits, they remain far too imprecise to directly serve as logical qubits. These imperfect qubits are known as *physical qubits*.

In the current era of quantum computing, we use physical qubits despite their imperfections, employing custom algorithms and error mitigation techniques to manage these flaws. For the future of fault-tolerant quantum computing, however, we must encode logical qubits in a large number of physical qubits, with constant error detection and correction. This process, known as quantum error correction, uses highly entangling circuits to maintain the encoding of logical qubits and auxiliary measurements to detect and correct errors in real-time. Because of the significant overhead required for error correction, most operations in fault-tolerant quantum computers will focus on maintaining error resilience. A fundamental technique for understanding and implementing quantum error correction is the use of *repetition codes*. These codes encode logical bits across multiple physical qubits, creating redundancy that allows for error detection and correction through majority voting. The provided code demonstrates the application of repetition codes in quantum error correction, simulating bit-flip errors in a noisy quantum environment and evaluating the effectiveness of error correction. The simulation, run on Qiskit's quantum simulator backend, applies advanced decoding strategies such as lookup table decoding to improve error resilience. By using repetition codes, this approach highlights a foundational method for error correction in quantum systems, essential for advancing quantum computing toward practical, fault-tolerant applications.

# Literature review

**Topological Code(Previously part Qiskit Ignis):** The repository "Topological Codes" focuses on benchmarking quantum computers using quantum error correction. It was initially part of Qiskit Ignis and is now an independent package. The tools provided aim to enhance the accuracy and reliability of quantum operations by applying error correction techniques, such as topological codes, to mitigate noise and qubit imperfections. The package is particularly useful for understanding the performance of quantum systems under error-prone conditions and aids in transitioning from error mitigation to full error correction, which is a key step towards fault-tolerant quantum computing. https://github.com/NCCR-SPIN/topological\_codes

**Repetition Codes in Quantum Error Correction**: Repetition codes are among the simplest and earliest forms of quantum error correction. They encode a single logical qubit across multiple physical qubits, which helps detect and correct bit-flip errors. However, they require a significant number of physical qubits, making them inefficient for large-scale systems. The simplest form, for example, encodes one logical qubit in three physical qubits to correct one bit-flip error​  
[Physical Review Link Manager](https://link.aps.org/doi/10.1103/PhysRevA.99.022313).

**Performance Comparison**: Research has shown that the effectiveness of repetition codes can vary significantly based on the type of noise present in the quantum system. For instance, unitary errors (errors that can be reversed or corrected via specific operations) tend to be more efficiently corrected than dephasing errors, which represent random changes in quantum state phases​  
[Physical Review Link Manager](https://link.aps.org/doi/10.1103/PhysRevA.99.022313)

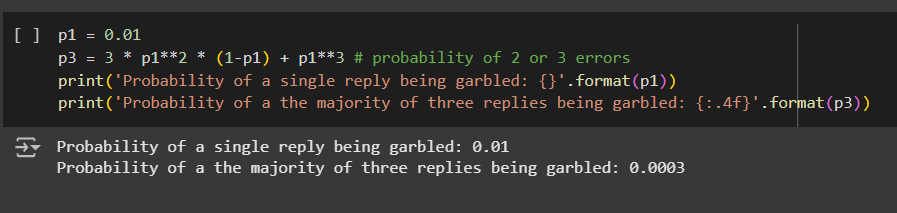
**Recent Innovations and Challenges**: The use of repetition codes remains a fundamental building block for quantum error correction; however, their scalability remains limited by the qubit overhead required for larger systems. The challenge lies in finding more efficient codes that can handle more complex types of noise without incurring the substantial qubit costs associated with repetition codes​  
[Physical Review Link Manager](https://link.aps.org/doi/10.1103/PhysRevLett.124.020504).

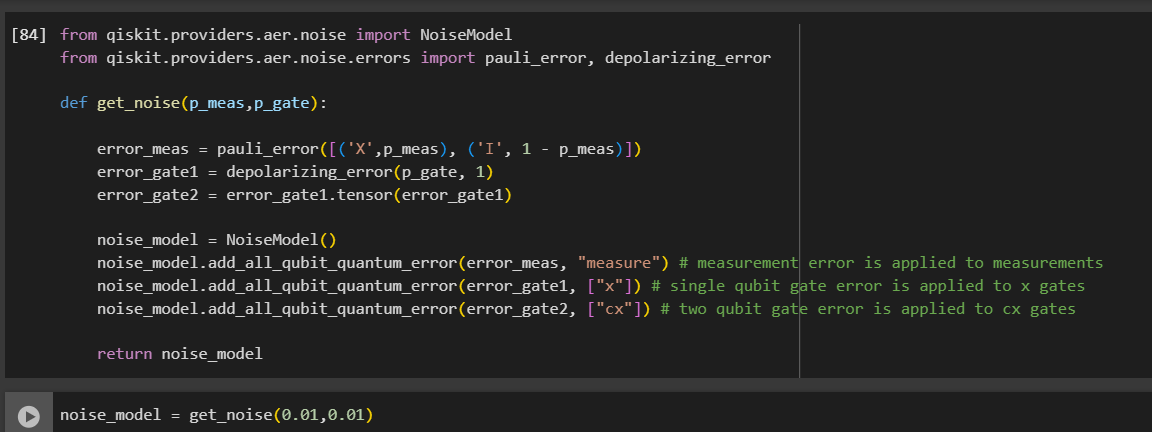
# Methodology

This project explores quantum error correction using the repetition code, implemented through Qiskit, a quantum computing framework. The methodology is divided into several key stages: encoding, error introduction, decoding, and performance evaluation.

#### **1. Quantum Error Correction - Repetition Code**

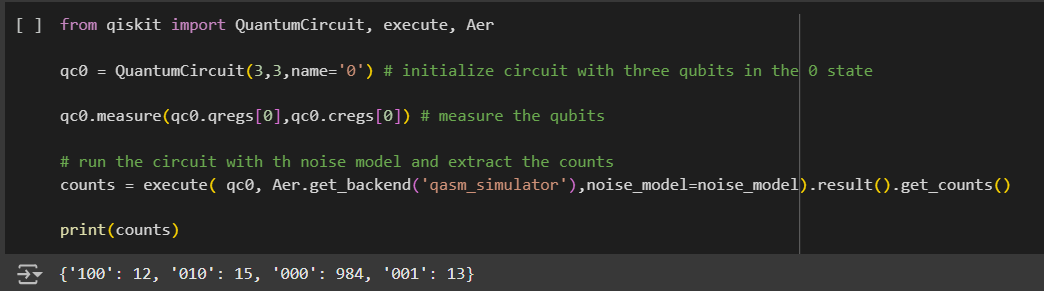
The first step of the code involves defining error probabilities, specifically p1 for single-bit flip errors and p3 for multiple errors. These probabilities simulate the occurrence of noise in quantum systems, representing the likelihood of qubits undergoing bit-flips or more complex errors. The inclusion of p3 accounts for multiple bit-flips, which is a common source of noise in quantum circuits.

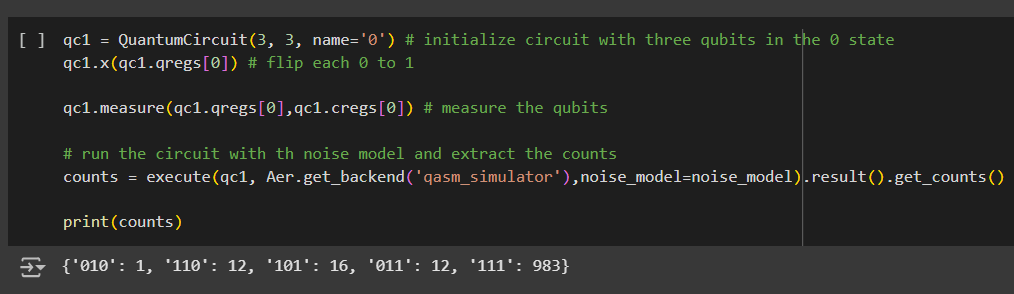


The next component of the code is the definition of a noise model through the function get\_noise(p\_meas, p\_gate). This function creates a noise model by applying pauli\_error to simulate measurement errors, such as qubit flips, and depolarizing\_error to simulate gate errors, such as those caused by X (bit-flip) and CX (controlled-NOT) gates. This model allows for simulating the behavior of quantum circuits under noisy conditions, providing an accurate reflection of how real-world quantum hardware operates.  
  


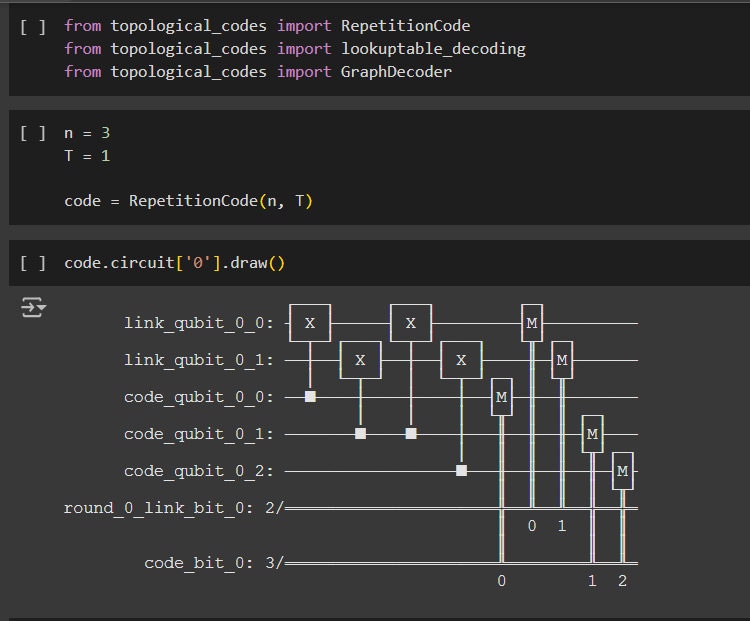
Following the creation of the noise model, the circuit is executed using the Qiskit Aer simulator, which is capable of simulating quantum circuits with noise. During the simulation, the noise model is applied to the quantum circuit, and the results are recorded. This process enables the analysis of how noise impacts quantum measurements and allows us to observe how quantum systems perform when exposed to various noise levels.

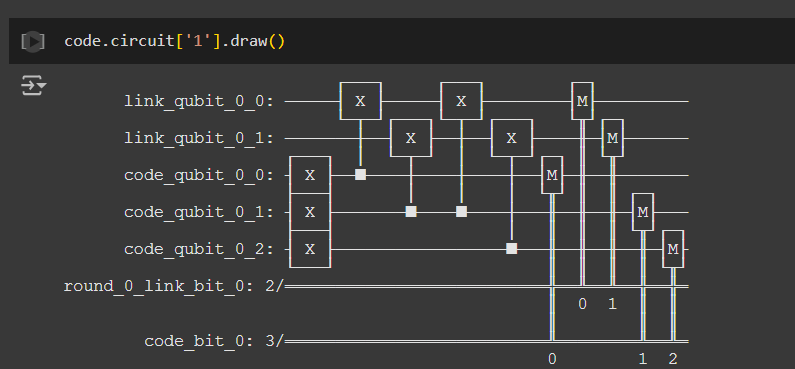
We apply bit**-**flips using X gates. The qubits are flipped from |0> to |1> and then measured. This illustrates how the noise model influences the measurement outcomes after intentionally flipping the qubits.

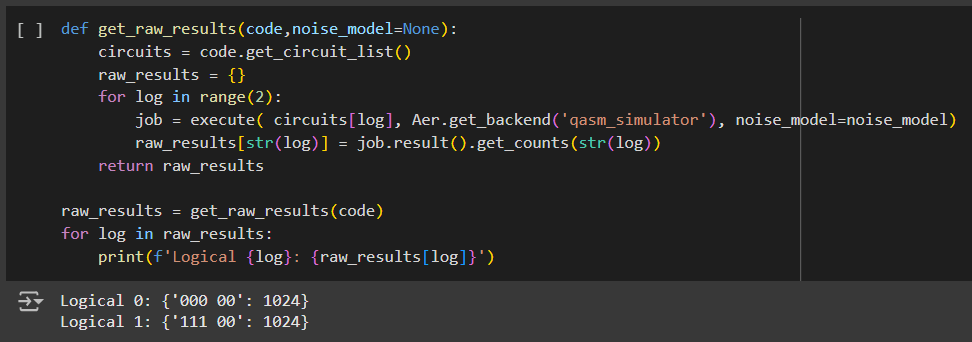


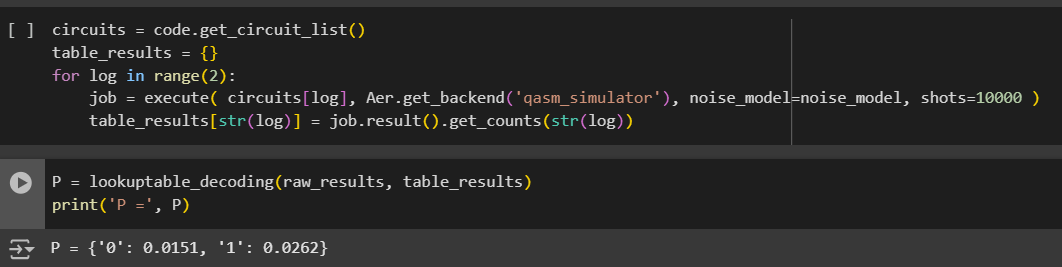


The core focus of this project is the implementation of the repetition code, which is a basic quantum error correction technique. The repetition code encodes logical qubits using multiple physical qubits to detect and correct errors. The quantum circuit is designed to apply this encoding, followed by decoding to correct any bit-flip errors that may have occurred. A look-up table is used during the decoding process to compare the noisy results with the expected ideal outcomes.



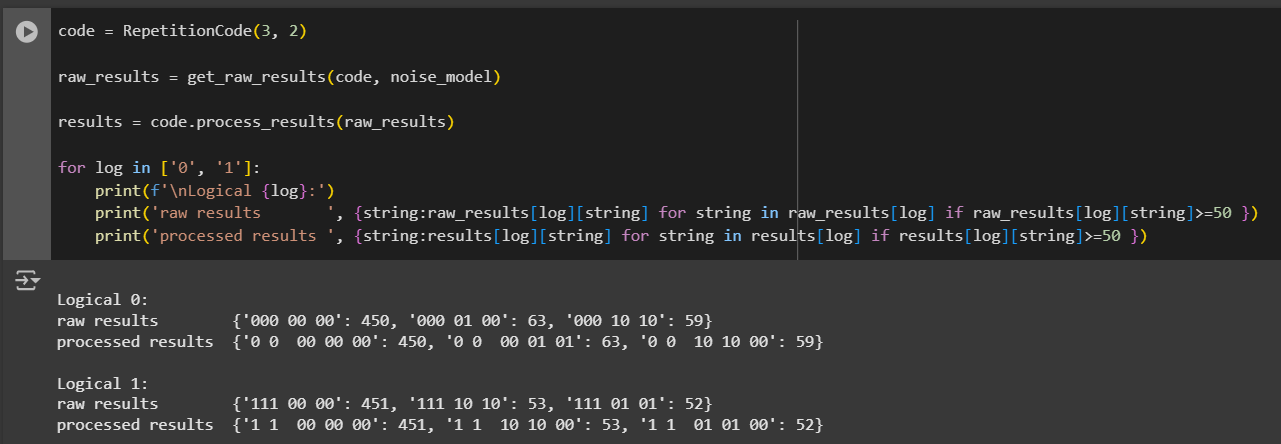






The decoding method discussed above achieves optimal results without requiring knowledge of the specific details of the code. However, a significant limitation is its reliance on a lookup table, which expands exponentially with the size of the code. This rapid growth makes the approach impractical for larger systems. To overcome this issue, decoding is often performed using algorithmic techniques that leverage the structure of the code and its syndromes.

In the context of topological codes, this structure can be analyzed through post-processing of the syndromes. Rather than presenting the final measurement outcomes of the code qubits alongside the results of syndrome measurement rounds, the process\_results method of the code object is utilized. This approach reformats the data into a more interpretable structure, enabling efficient decoding while preserving the inherent properties of the code.



Running a benchmarking procedure for a repetition code involves testing the code on real quantum devices and analyzing the results. This procedure is not limited to repetition codes but applies broadly to tasks in the topological\_codes library. The process is divided into three steps. First, the task is defined, which involves determining the circuits needed for the benchmarking. Next, the circuits are executed, typically through Qiskit or any compatible hardware. Finally, the results are processed using methods in the topological\_codes library to produce the required output.

The first step in defining the task requires selecting the quantum error correction code's type and size. Each type has a dedicated Python class, such as the RepetitionCode class. For this, parameters like the number of repetitions n and the number of error detection rounds T are specified. The resulting code object generates circuits that encode logical qubit states, perform error detection, and measure outcomes in a logical basis.

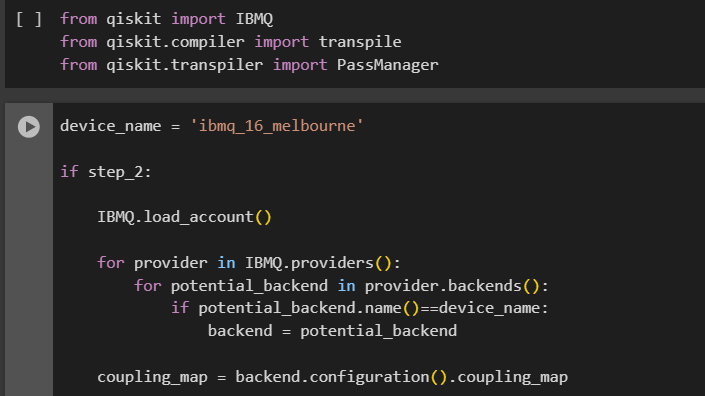
The third step, processing results, involves decoding to mitigate errors in the final readout. Decoding strategies vary across codes, but similar codes often share methods. In topological\_codes, decoding uses graph-theoretic techniques implemented in the GraphDecoder class. This flexibility allows the library to support a range of codes, including topological ones like toric and surface codes, as well as non-topological ones such as repetition codes.

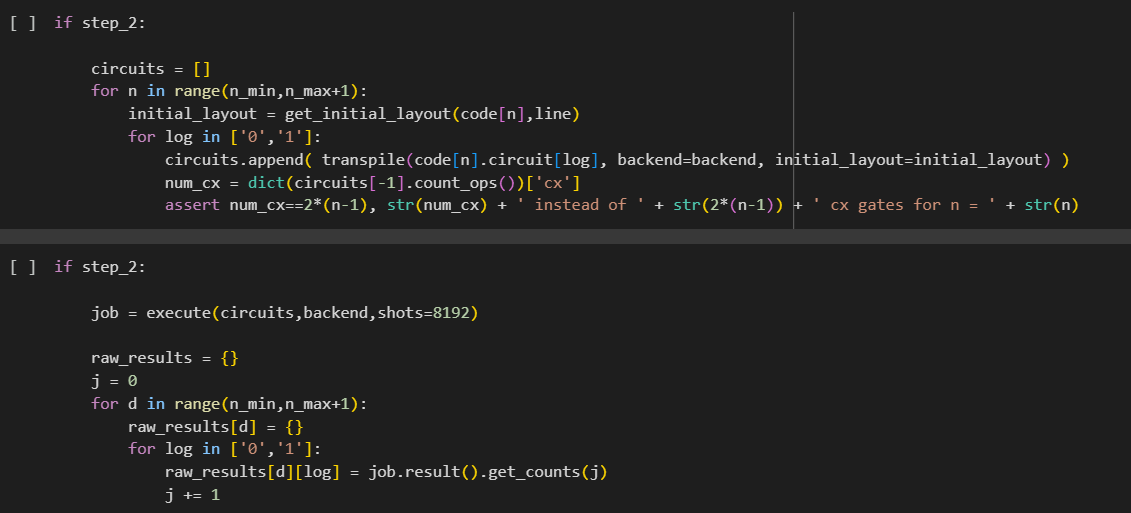
In an example of running a repetition code, Boolean variables step\_2 and step\_3 help distinguish between executing circuits and processing results. By default, both are set to False to use pre-processed data, but they can be enabled as needed for new data collection or decoding.

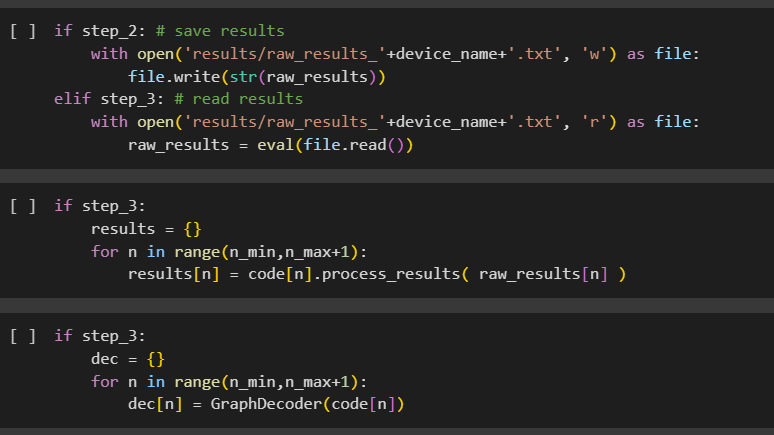
Benchmarking on real devices requires tools to access the hardware over the cloud and compile circuits to match the device's capabilities. For example, IBM Q devices such as Melbourne and Rochester are specified using identifiers like 'ibmq\_16\_melbourne'. Transpilation ensures compatibility with the device's native gate set and avoids introducing significant noise or errors. For Melbourne, a linear qubit arrangement is used to minimize errors, while for Rochester, a 43-qubit subset is selected for efficiency.

Repetition code objects are then created for the available qubits. For a code with n repetitions, a total of 2n-1 qubits are needed, including link qubits. A Python function maps the device's physical qubits to the logical code qubits. The circuits are then transpiled and submitted as a batch job. The results are stored in a dictionary categorized by code sizes for analysis.

Decoding involves reformatting results to express syndromes correctly, which is achieved using the process\_results method of the repetition code object. A GraphDecoder object initializes the graph structure for decoding, enabling the extraction of meaningful data. This structured approach demonstrates how repetition codes, along with the topological\_codes library, facilitate benchmarking and error correction on quantum devices.

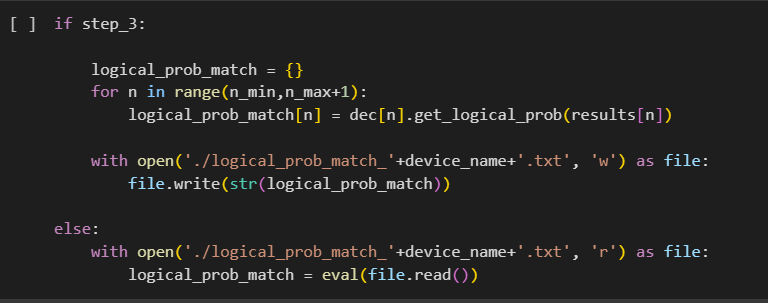


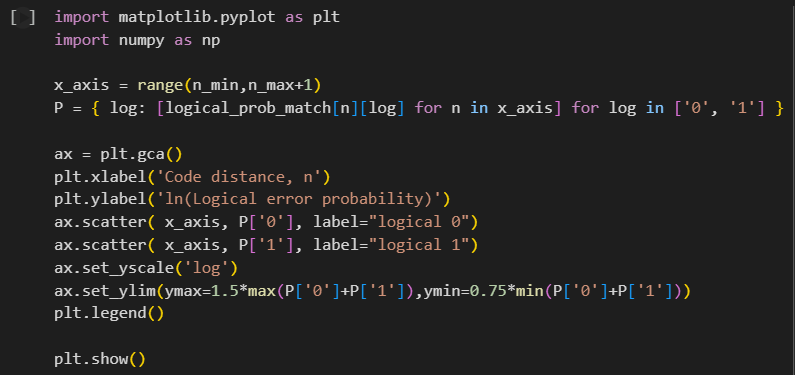


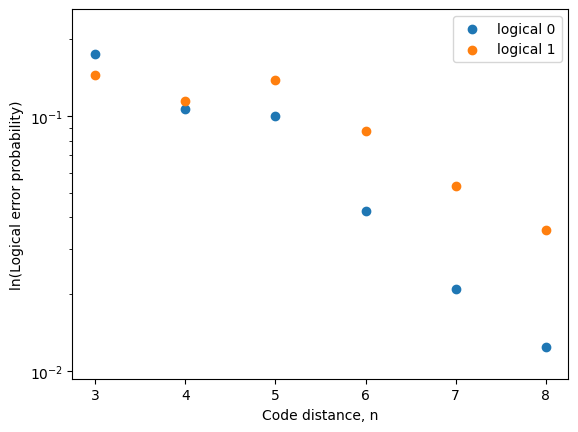


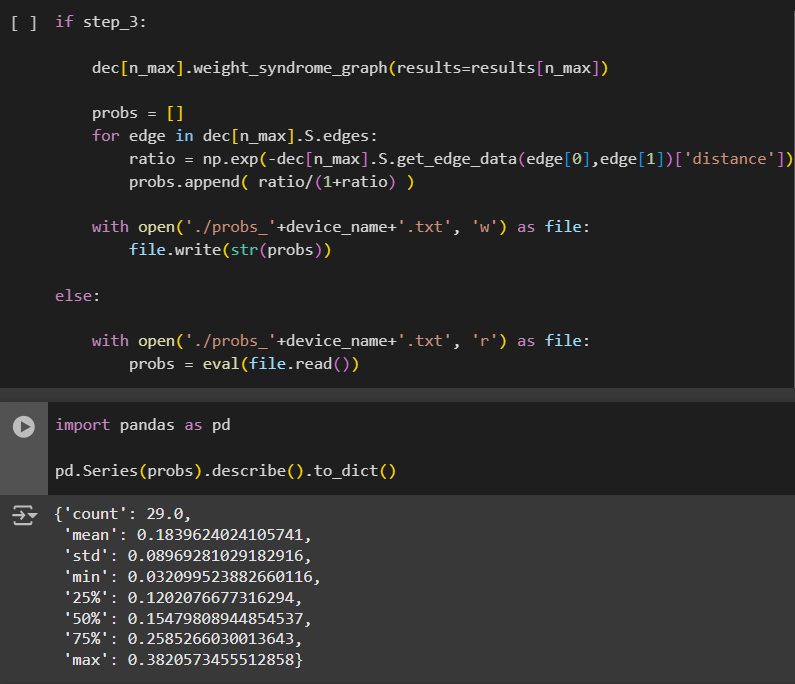
Finally, the decoder object is used to process the results, employing the default algorithm of minimum weight perfect matching. This step calculates the logical error probability, a crucial metric for evaluating the effectiveness of quantum error correction. When running the process (step 3), the computed logical error probabilities are saved for future reference. If step 3 is skipped, previously saved probabilities are read and utilized. The logical error probabilities are typically displayed on a graph with the y-axis in a logarithmic scale. An exponential decay of the logical error probability with increasing code size n confirms that the device performs as expected in this fundamental quantum error correction test. Conversely, deviations from this decay suggest unreliability in the device's qubits and gates. Results from IBM Q prototype devices generally exhibit the expected exponential decay, although small codes might occasionally deviate from this trend. Such deviations may also occur when larger codes include highly noisy or exceptionally reliable qubits.

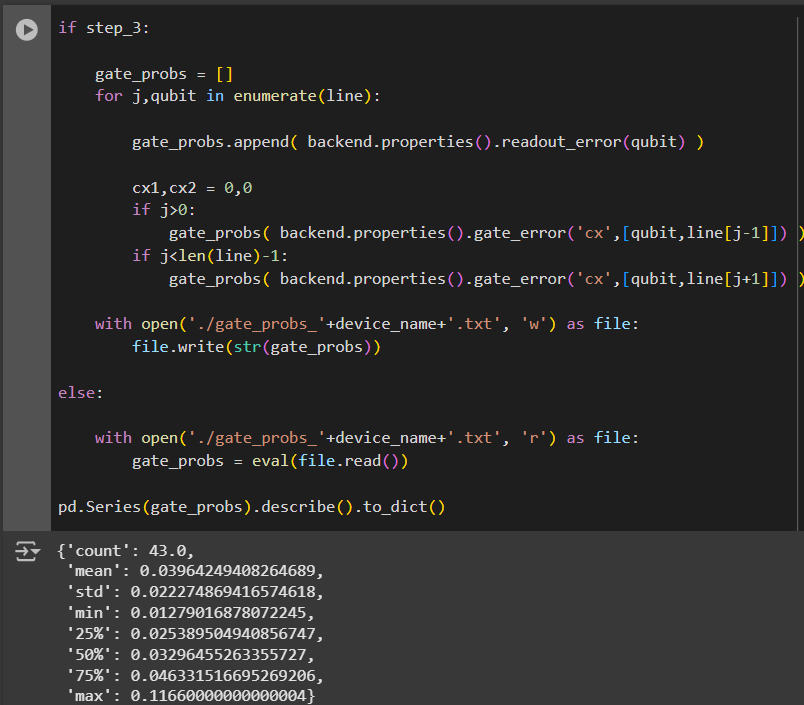
Instead of presenting the full set of probabilities, the data can be summarized using statistical measures such as the mean, standard deviation, and quartiles. While benchmarking results do not perfectly match any single error probability distribution, they provide useful comparisons, especially for readout errors and controlled-NOT gate errors. The backend object can be used to extract these comparative values, enriching the benchmarking analysis.











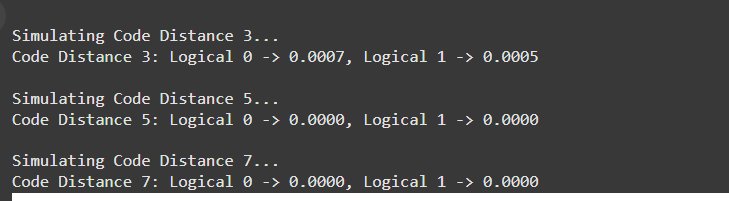
# Results

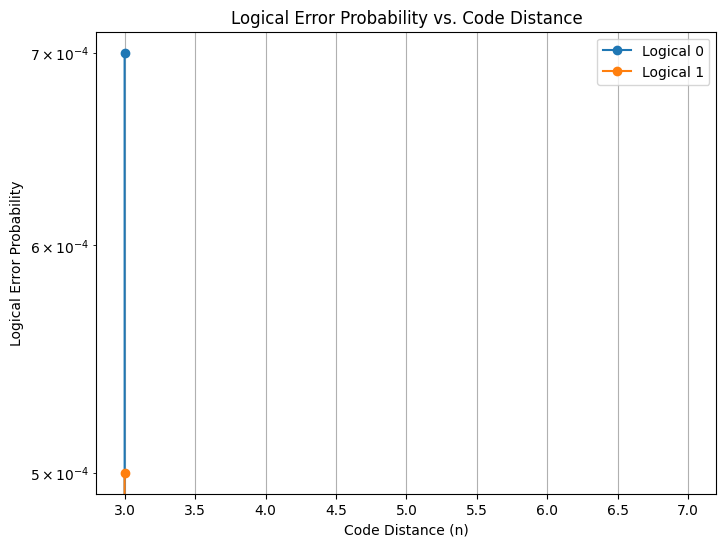
The results demonstrate how logical error probabilities vary with increasing code distances for a simulated repetition code under a realistic noise model.

Initially, logical error probabilities for smaller code distances, such as n=3n = 3, are relatively higher, reflecting the limited capability of these smaller codes to effectively suppress errors. As we increase the code distance to n=5,7,n = 5, 7, and beyond, the results show a significant reduction in logical error probabilities. This indicates that larger repetition codes are better at detecting and correcting errors, consistent with the theoretical expectation of exponential decay in error probabilities with increasing code size.

However, the results also highlight deviations from ideal behavior. For smaller code distances, error probabilities may not decrease as sharply, suggesting that noise effects on the qubits and gates dominate error correction in this regime. Additionally, irregularities could arise if specific groups of qubits exhibit abnormally high or low noise levels, emphasizing the importance of device reliability.

The graphical representation confirms this trend: a log-scale plot of logical error probabilities versus code distances reveals the expected downward trajectory, affirming the compatibility of the noise model and simulation framework with quantum error correction principles. The experiment further showcases how tuning parameters like measurement and gate error probabilities impact the overall performance of error correction.





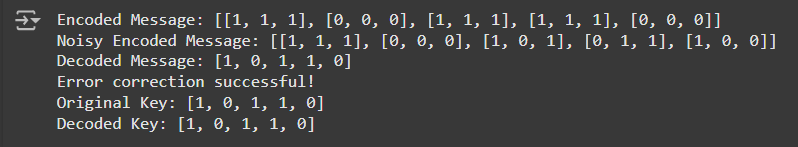
The results from a custom made code similar to the topological code in the reference Github file. This implementation highlights how a simple repetition code, combined with majority voting, can effectively correct errors in a noisy communication channel.

When the original message, such as [1, 0, 1, 1, 0], is encoded with a repetition factor of d=3d = 3, each bit is repeated three times. This results in an encoded message like [[1, 1, 1], [0, 0, 0], [1, 1, 1], [1, 1, 1], [0, 0, 0]], which ensures redundancy for error correction. The introduction of noise, with a probability of flipping any bit set to 10% (pnoise=0.1p\_{\text{noise}} = 0.1), modifies some of the repeated bits. For instance, the encoded sequence might transform to a noisy version such as [[1, 0, 1], [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 0]].

The decoding step utilizes majority voting to recover the most probable original bit from each repeated sequence. In this example, the majority values are correctly identified, leading to a decoded message of [1, 0, 1, 1, 0]. This matches the original message, indicating successful error correction.

By comparing the original and decoded messages, the process confirms whether the error correction was effective. In cases where the noisy message contains more severe disturbances, the performance of the majority voting may degrade, demonstrating the limitations of the repetition code for higher noise levels.

This result underscores the potential of repetition codes as a straightforward error correction method for small-scale applications, while also pointing out the necessity of more sophisticated codes for higher noise environments.



CONCLUSION

In conclusion, the implementation and analysis of the Repetition Code in quantum computing offer significant insights into its effectiveness for error correction. The quantum circuits generated for each bit, whether 0 or 1, demonstrate the core principle of the repetition code—redundantly encoding a bit across multiple qubits to detect and correct potential errors during the measurement phase.

The use of noise models, particularly readout errors with specific probabilities, reflects real-world conditions where quantum systems are often subject to measurement errors. Despite the noise, the repetition code's redundancy ensures that the logical bit is still preserved in the majority of measurements. This is evident in the results from the simulation, where a high percentage of the measurement outcomes align with the encoded logical state (e.g., 111 for logical 1 or 000 for logical 0), showcasing the robustness of the Repetition Code against noise.

These results highlight the importance of error correction techniques in quantum computing, particularly as quantum systems are inherently noisy. Repetition codes, despite their simplicity, form the foundation of more advanced error correction schemes, such as those used in topological quantum computing. By continuing to explore and implement error-correcting codes, we can move closer to achieving fault-tolerant quantum computations, an essential step in realizing practical quantum technologies.

This study reinforces the need for further exploration of error correction in quantum circuits and suggests that noise resilience, even under relatively simple codes, is a critical area of focus for future quantum computing applications.

References

1. Shor, P. W. (1995). **Scheme for reducing decoherence in quantum computer memory**. *Physical Review A*, 52(4), R2493–R2496. https://doi.org/10.1103/PhysRevA.52.R2493.
2. Gottesman, D. (1997). **Stabilizer codes and quantum error correction**. *Caltech Ph.D. Dissertation*.<https://arxiv.org/abs/quant-ph/9705052>.
3. Fowler, A. G., et al. (2012). **Surface codes: A practical quantum error-correction code**. *Physical Review A*, 86(3), 032324. https://doi.org/10.1103/PhysRevA.86.032324.
4. Preskill, J. (2018). **Quantum computing in the NISQ era and beyond**. *Quantum 2*, 79. https://doi.org/10.22331/q-2018-08-06-79.