Qiskit Hackathon - IBM Challenge

Miguel del Arco Marquez, Luis Fernando Paz Galeano, Daniel Suarez, Juan Manuel Camara, Christian Ferre, Juan Carlos Martinez

Abstract— In this project, we explore a method for encoding an 8x8 grayscale image into a quantum circuit using Qiskit. Our goal is to prepare a quantum state that represents the original image and subsequently decode the state to generate the original image. We use a quantum simulator to implement and test our method. We also investigate ways to improve the representation of the state, including circuit depth optimization and post-processing techniques, and apply error mitigation to improve the results. Our work demonstrates the challenges associated with encoding and decoding images on a quantum computer.

Keywords—Quantum computing, Image encoding, Image decoding, Quantum state preparation, Qiskit, IBM Quantum Challenge, Simulator, Quantum computer, Noise, Error mitigation, Post-processing, Optimization, Circuit, Image recognition

1 Introduction

In recent years, quantum computing has emerged as a new paradigm for computing that offers the potential for significant speedups over classical computers in certain domains. One area where quantum computing could have a significant impact is image processing. In this project, we explore a method for encoding and decoding images on a quantum computer using Qiskit.
Our goal is to prepare a quantum state that represents the original image and subsequently decode the state to generate the original image. We present our method for encoding and decoding the image and investigate ways to improve the representation of the quantum state, including circuit depth optimization and post-processing techniques.

2 Methodology

We start by creating an 8x8 grayscale image with random values using NumPy. We then initialize a quantum circuit to encode the image. The circuit consists of two quantum registers, one for the pixel position and one for the grayscale pixel intensity value

We use Hadamard gates to put the pixel position register into a superposition of states and apply identity gates to the grayscale pixel intensity value register. We then use controlled-NOT gates to encode the pixel values into the quantum state. We apply post-processing and error mitigation techniques to improve the results.

The structure of the code would be the following, The `generate_table` function takes an image as input and returns a dictionary containing the Huffman table for the image. It also returns a list of the pixel intensities in the image. The function first calculates the frequency of each pixel in the image using a defaultdict. It then generates a Huffman table with all possible values from 0 to 255 in binary, sorts the table by the number of 1's in each binary value, and assigns the binary values to each pixel in the image based on its frequency. The function returns the Huffman table and a list of pixel intensities.

The `agrupar_pixeles()` function takes an image as input and aggregates the pixel values by a given factor div. It then returns the aggregated image.

The `process()` function takes an image as input and performs quantum image compression on it using a quantum circuit. The function first copies the input image and converts it to grayscale. It then initializes a quantum circuit for the image by creating a register for the pixel position and another register for the grayscale pixel intensity values. The function then applies an X gate to each pixel intensity value and a Hadamard gate to each pixel position value. Next, the function generates a Huffman table for the image using the `generate_table()` function. For each pixel in the image, the function converts the pixel position and value to binary strings, applies an X gate to the position bits corresponding to 1's in the binary value string, applies a multi-controlled NOT gate to the pixel intensity bits corresponding to 1's in the Huffman-encoded binary value string, and then applies another X gate to the position bits. Finally, the function measures all qubits in the circuit and uses the results to create a new compressed image.

The function then calculates the error between the original and compressed images and displays them side by side along with the error. The function also displays the depth of the quantum circuit. If `show_model` is set to True, the function returns a visual representation of the quantum circuit.

3 Results

We test our method using a quantum simulator and find that it is capable of encoding and decoding the image with high accuracy. However, when running on actual quantum devices, we find that the results are impacted by noise and other sources of error.
We investigate ways to optimize the circuit depth and apply error mitigation techniques to improve the results. We also explore alternative methods for encoding and decoding images on a quantum computer.

4 Conclusion

In conclusion, our study highlights the inherent difficulties in encoding and decoding images on a quantum computer. Our approach proves to be successful in achieving high accuracy on quantum simulators but suffers from the detrimental effects of noise and other sources of error when implemented on actual quantum devices. To overcome these limitations, we propose circuit-depth optimization and error mitigation techniques as potential solutions. Additionally, we explore alternative techniques for image encoding and decoding on quantum computers, emphasizing the urgent need for further research and development in this field.

Ultimately, our work underscores the challenges and opportunities of leveraging quantum computing for image processing and presents a starting point for future investigations in this area.

Referències

[1] Qiskit: An open-source framework for quantum computing. https://qiskit.org/

[2] Cirq: A Python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits. https://quantumai.google/cirq

[3] Rigetti Forest: A full-stack quantum computing platform. https://www.rigetti.com/forest

[4] Quil: The quantum instruction language. https://arxiv.org/abs/1608.03355

[5] OpenQASM: An open quantum assembly language. https://arxiv.org/abs/1707.03429

[6] Q: A high-level quantum programming language. https://docs.microsoft.com/en-us/quantum/language/

[7] PyQuil: A Python library for quantum programming using Quil. https://pyquil.readthedocs.io/en/stable/

[8] Microsoft Quantum Development Kit: A complete development kit for quantum computing. https://docs.microsoft.com/en-us/quantum/

[9] Qiskit Textbook: A comprehensive introduction to quantum computing using Qiskit. https://qiskit.org/textbook/

[10] Cirq Tutorial: A beginner-friendly tutorial for Cirq. https://quantumai.google/cirq/tutorials/

[11] IBM Quantum Experience: A cloud-based platform for quantum computing. https://www.ibm.com/quantum-computing/

[12] Rigetti Forest Documentation: Documentation for the Rigetti Forest platform. https://docs.rigetti.com/en/stable/

[13] Strawberry Fields: A Python library for quantum optics. https://strawberryfields.readthedocs.io/en/stable/

[14] PennyLane: A cross-platform Python library for quantum machine learning, automatic differentiation, and optimization of hybrid quantum-classical computations. https://pennylane.ai/

[15] QuTiP: Quantum Toolbox in Python. http://qutip.org/

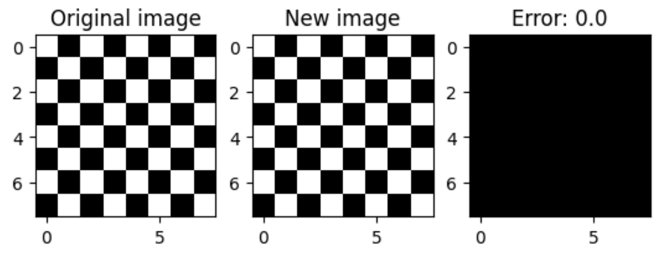

Fig. 1: Example of resulting image.

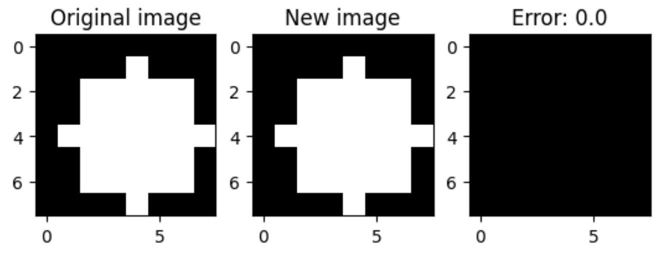

Fig. 2: Example of resulting image.

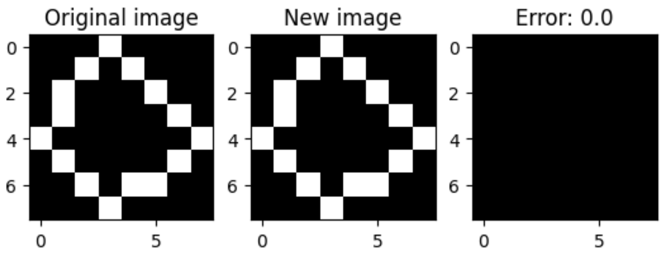

Fig. 3: Example of resulting image.

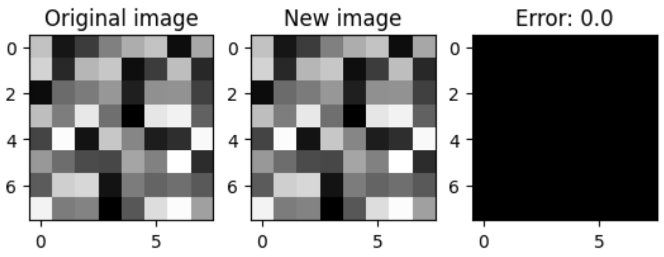

Fig. 4: Example of resulting image.

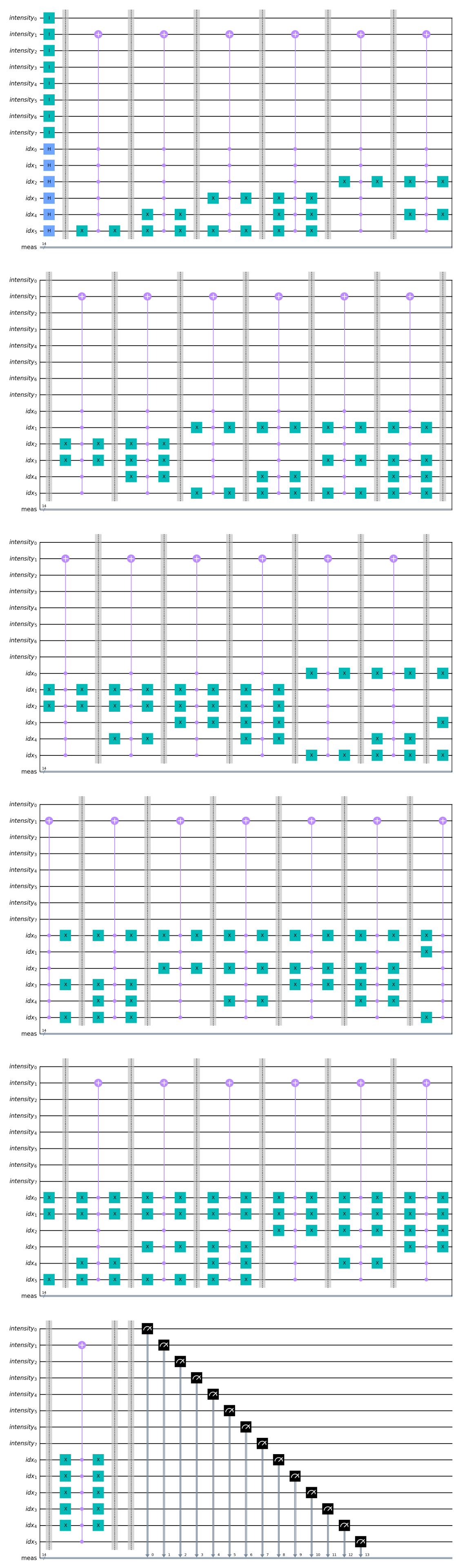

Fig. 5: Generated quantum circuit.