

LZW-Driven Medical Image Compression



Mini Project submitted in partial fulfillment of the requirement for the award of the
degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Dr M.Prasanth
Assistant Professor

By

CH SRICHARAN	(21R11A05B7)
SADANAND GOUD	(21R11A05E2)
SATYAM DAS	(21R11A05E5)



Department of Computer Science and Engineering
Accredited by NBA

Geethanjali College of Engineering and Technology
(UGC Autonomous)

(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

August-2024

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA



CERTIFICATE

This is to certify that the B.Tech Mini Project report entitled **“LZW-Driven Medical Image Compression”** is a bonafide work done by **Chebolu Sricharan (21R11A05B7), Sadanand Goud (21R11A05E2), Satyam Das (21R11A05E5)**, in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in **“Computer Science and Engineering”** from Jawaharlal Nehru Technological University, Hyderabad during the year 2023-2024.

Internal Guide

Dr M. Prashanth

Assistant Professor

HOD – CSE

Dr A SreeLakshmi

Professor

External Examiner

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA



DECLARATION BY THE CANDIDATE

We, **Chebolu Sricharan, Sadanand Goud, Satyam Das**, bearing Roll Nos. **21R11A05B7, 21R11A05E2, 21R11A05E5**, hereby declare that the project report entitled “**LZW-Driven Medical Image Compression**” is done under the guidance of **Mr. M.Prasanth, Assistant Professor**, Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by me/us in **Geethanjali College of Engineering and Technology** and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Chebolu Sricharan (21R11A05B7)

Sadanand Goud(21R11A05E2)

Satyam Das (21R11A05E5)

Department of CSE,
Geethanjali College of Engineering and Technology,
Cheeryal.

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. A. Sree Lakshmi, Professor, Head of Department** of Computer Science, Geethanjali College of Engineering and Technology, Cheeryal, whose motivation in the field of software development has made us to overcome all hardships during the course of study and successful completion of project.

We would like to express our profound sense of gratitude to all for having helped us in completing this dissertation. We would like to express our deep-felt gratitude and sincere thanks to our guide **Dr. M. Prasanth, Assistant Professor**, Department of Computer Science, Geethanjali College of Engineering and Technology, Cheeryal, for his skillful guidance, timely suggestions and encouragement in completing this project successfully.

We would like to express our sincere gratitude to our **Principal Prof. Dr. S. Udaya Kumar** for providing the necessary infrastructure to complete our project. We are also thankful to our Secretary **Mr. G. R. Ravinder Reddy** for providing an interdisciplinary & progressive environment.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set goals.

Chebolu Sricharan (21R11A05B7)

Sadanand Goud (21R11A05E2)

Satyam Das (21R11A05E5)

ABSTRACT

Medical imaging techniques produce large volumes of image data, posing significant storage and transmission challenges. This project is aimed to develop an efficient medical image compression system using the LZW(Lempel-Ziv-Welch) algorithm. The LZW algorithm is a very common compression technique. It is lossless, meaning no data is lost when compressing. The main aim of using the LZW algorithm is to optimize image quality and compression ratios. This allows to reduce the image file size, making it easier to store and transmit. This is useful for telemedicine applications, enabling faster image sharing and diagnosis. The LZW algorithm works by finding repeated patterns in the image data and replacing them with a reference to the previous occurrence of the pattern. It reads a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. LZW looks for repeated patterns in an image, like a row of identical pixels. When it finds a repeated pattern, it replaces it with a short code. The code is stored instead of the original code and the LZW continues finding and replacing patterns until the entire image is compressed. Compression of medical data before storage is very important to save storage space. The Stored data can be accessed online with very less bandwidth. The proposed method would be very useful in developing telemedicine applications .LZW compression used in the proposed method can be used in compressing the medical data better than run length coding and huffmann coding.

LIST OF FIGURES

S.No	Figure Name	Page No
1	Usecase Diagram	10
2	Sequence Diagram	11
3	Activity Diagram	13
4	Upload File Page	24
5	Upload Result Page	24
6	Decompressed Download Page	25
7	Original Image	25
8	Decompressed Result	26
9	Upload Result	26
10	Downloaded Image	27
11	Encryption and Decryption command	27

TABLE OF CONTENTS

S.No	Contents	Page No
	Abstract	v
	List Of Figures	vi
1	Introduction	
	1.1 About the Project	1
	1.2 Objectives	2
2	System Analysis	
	2.1 Existing System	3
	2.2 Proposed System	4
	2.3 Feasibility Study	5
	2.4 Scope of project	6
	2.5 System Configuration	7
3	Literature Overview	
	3.1 Literature Review	8
4	System Design	
	4.1 System Architecture	10
	4.2 UML diagrams	12
5	Implementation	
	5.1 Working/Implementation	15
	5.2 Sample Code	18
6	Testing	
	6.1 Testing	22
	6.2 Test Cases	22
7	Output Screens	24
8	Conclusion	
	8.1 Conclusion	28
	8.2 Future Enhancement	28
9	Bibilography	
	9.1 References	30
10	Appendices	31

1. INTRODUCTION

1.1 ABOUT THE PROJECT

The project titled "LZW-Driven Image Compression for Efficient Telemedicine" focuses on developing a system to efficiently compress medical images using the LZW (Lempel-Ziv-Welch) algorithm. The system is designed to address the storage and transmission challenges faced in telemedicine, where large volumes of image data from medical imaging techniques need to be processed. The LZW algorithm is particularly effective as it is a lossless compression technique, ensuring that no data is lost during the compression process. By reducing the image file size while maintaining quality, the project aims to make image storage and transmission more efficient in telemedicine applications

1.2 OBJECTIVES

- **Efficient Compression:** Develop an effective image compression system using the LZW algorithm to handle large volumes of medical images, focusing on high compression ratios without compromising image quality.
- **Telemedicine Optimization:** Enhance the speed and efficiency of telemedicine by enabling faster image sharing and diagnosis through the reduction of file sizes.
- **Quality Maintenance:** Ensure that the reconstructed images maintain high quality, even after compression, for accurate medical diagnostics.
- **Application Testing:** Test the algorithm with medical images from various modalities, such as MRI and CT scans, to validate performance and effectiveness.
- **Comparative Analysis:** Demonstrate how LZW compression performs better than other techniques like Run-Length Coding and Huffman Coding for medical data compression.
- **Resource Efficiency:** Minimize storage requirements and bandwidth usage in telemedicine systems by compressing data before storage and enabling easy access onl

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

In the existing medical imaging systems, large volumes of image data are generated from modalities like MRI, CT scans, and X-rays. These images are typically stored and transmitted without significant compression, leading to challenges in terms of storage space and transmission speed. Current methods often use basic compression techniques such as Run-Length Encoding (RLE) or Huffman Coding, which are either lossy or less efficient in compressing complex medical images.

These limitations result in:

1. **Large Storage Requirements:** The uncompressed or minimally compressed images occupy significant storage space.
2. **Slow Transmission:** The large file sizes lead to delays in transferring medical data, especially in telemedicine applications where timely diagnostics are critical.
3. **Loss of Image Quality:** In cases where lossy compression methods are used, the quality of medical images is compromised, which may affect accurate diagnosis.

2.2 PROPOSED SYSTEM

The proposed system introduces an LZW-driven image compression method specifically designed for medical images used in telemedicine. The LZW (Lempel-Ziv-Welch) algorithm is a lossless compression technique that optimizes both the compression ratio and image quality.

Key features of the proposed system include:

1. **Efficient Compression with LZW:** The system identifies repeated patterns within the image data and replaces them with shorter codes. This allows for significant reduction in file size while preserving the original image quality.
2. **Enhanced Telemedicine Capabilities:** The reduced file size facilitates faster transmission of images over limited bandwidths, making it suitable for real-time telemedicine applications.
3. **Quality Retention:** Since LZW is a lossless algorithm, the decompressed images retain their original quality, ensuring that critical medical details are not lost.
4. **Improved Performance:** The proposed system is tested across different imaging modalities like MRI and CT scans, showing better compression performance compared to existing methods like Run-Length Encoding and Huffman Coding.
5. **Scalability for Large Data Sets:** The system is designed to handle and compress large volumes of medical images efficiently, making it suitable for modern healthcare environments that generate vast amounts of imaging data.

2.3 FEASIBILITY STUDY

2.3.1 Details

The study confirms the feasibility of LZW-driven compression for medical images, offering efficient storage and transmission without compromising image quality. With a lossless compression ratio of up to 3:1, it meets diagnostic requirements while reducing storage needs and transmission times, making it a viable solution for telemedicine applications.

2.3.2 Impact on Environment

LZW-driven medical image compression reduces storage space and transmission bandwidth, thereby decreasing the energy consumption associated with data storage and transfer. This leads to a lower carbon footprint in healthcare IT infrastructure. However, the environmental impact depends on the efficiency of the algorithm and the scale of its deployment. Overall, it can contribute positively to environmental sustainability by optimizing resource usage.

2.3.3 Safety

The LZW-driven image compression project prioritizes safety in telemedicine by ensuring data integrity, patient privacy, and compliance with healthcare standards. It incorporates encryption, secure transmission, error handling, and ethical considerations to protect sensitive medical information and maintain trust in telemedicine services.

2.3.4 Ethics

The project focuses on using LZW-driven image compression for efficient telemedicine, aiming to enhance medical image storage and transmission. Ethically, the project could impact patient confidentiality, as handling medical images requires stringent privacy protections to prevent unauthorized access. The use of compression techniques, while beneficial for efficiency, must ensure that image quality remains sufficient for accurate diagnosis, avoiding potential harm to patients. Furthermore, fairness in access to telemedicine services should be considered, ensuring that this technology does not inadvertently widen the gap between those with and without access to digital healthcare.

2.3.5 Cost

The cost involved in the LZW-driven image compression project for telemedicine primarily includes the expenses related to software development, testing, and potential cloud storage or computing resources for handling and storing large volumes of medical images. Additionally, costs may be incurred for acquiring or accessing medical imaging data for testing and validation, as well as for any necessary hardware upgrades or maintenance. Given that it is an academic project, costs might be minimized through the use of open-source tools and institutional resources.

2.3.6 Type

This project is a software development and research project focused on medical image compression for telemedicine applications. Specifically, it involves developing and implementing a system based on the LZW (Lempel-Ziv-Welch) algorithm to efficiently compress medical images. The project falls under the category of computer science and engineering with applications in healthcare technology.

2.4 SCOPE OF THE PROJECT

The scope of this project includes developing an efficient image compression system using the LZW algorithm, specifically tailored for medical images such as MRI and CT scans. The project aims to optimize storage and transmission of large volumes of medical data in telemedicine, enhancing the speed and accessibility of remote diagnostics. By maintaining image quality while achieving high compression ratios, this system can significantly reduce the bandwidth required for transmitting medical images. The project's scope also extends to testing and validating the algorithm's effectiveness compared to other compression techniques like run-length coding and Huffman coding, making it potentially applicable in real-world telemedicine platforms. Future developments could include integrating this system with existing telemedicine software or extending it to other medical imaging modalities.

2.5 SYSTEM CONFIGURATION

The system configuration required for the LZW-driven image compression project for telemedicine may include the following:

1.Hardware:

- Processor: Intel Core i5 or higher
- RAM: 8 GB or more (preferably 16 GB for handling large image files)
- Storage: 256 GB SSD or higher
- GPU: Optional, but beneficial for faster processing of large datasets

2.Software:

- Operating System: Windows 10/11, Linux, or macOS
- Programming Language: Python or C++ for implementing the LZW algorithm
- IDE: Visual Studio Code, PyCharm, or similar
- Libraries: OpenCV, NumPy for image processing and data handling
- Version Control: Git for managing code versions
- Testing Tools: Jupyter Notebook or equivalent for algorithm testing and validation

3.Network:

- Internet connection with sufficient bandwidth for accessing online resources and telemedicine platforms.

3. LITERATURE OVERVIEW

3.1 LITERATURE REVIEW

The literature review for the LZW-driven image compression project in telemedicine involves exploring existing research and methodologies related to image compression, particularly in the context of medical imaging.

1. Medical Image Compression Techniques: Various methods for compressing medical images have been explored in the literature, including lossless techniques like Huffman coding, run-length encoding, and Lempel-Ziv-Welch (LZW) compression. Lossless compression is crucial in medical imaging to preserve the diagnostic quality of images. Studies have shown that while other techniques like JPEG and wavelet-based methods offer high compression ratios, they often result in some loss of data, which is unacceptable for medical diagnostics.

2. LZW Algorithm: The LZW algorithm, introduced by Abraham Lempel, Jacob Ziv, and Terry Welch, is a popular lossless data compression technique. It works by identifying repeating patterns within data and replacing them with shorter codes. Several studies have highlighted its effectiveness in compressing text and image files without loss of information. The literature suggests that LZW is particularly well-suited for applications requiring both high compression ratios and data integrity, making it a viable option for medical image compression.

3. Application in Telemedicine: With the growth of telemedicine, the need for efficient image compression has become more pronounced. Literature on telemedicine emphasizes the importance of fast and secure transmission of medical data, particularly in rural or underserved areas where bandwidth may be limited. Research indicates that efficient compression algorithms like LZW can play a critical role in enabling telemedicine by reducing the size of images transmitted over networks, thereby enhancing the speed and reliability of remote diagnostics.

4. Comparative Studies: Previous research comparing various compression algorithms for medical images indicates that while LZW may not always achieve the highest compression ratios compared to lossy techniques, its lossless nature and simplicity make it a preferred choice for medical applications. Studies comparing LZW with other lossless techniques like Huffman and run-length encoding have demonstrated its efficiency in maintaining image quality while providing reasonable compression.

5. Challenges and Considerations: The literature also discusses challenges in applying compression algorithms to medical images, such as the need to ensure that no clinically relevant information is lost during compression. Additionally, there is an ongoing debate about the balance between compression efficiency and computational complexity, especially when implementing these algorithms in real-time telemedicine systems.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The system architecture for the LZW-driven image compression project in telemedicine can be broken down into several key components. Below is a high-level overview of the architecture:

1. Input Layer:

- **Medical Image Acquisition:**
 - Sources: MRI, CT, X-ray, and other medical imaging modalities.
 - The system receives raw medical images from healthcare providers or imaging devices.

2. Preprocessing Layer:

- **Image Preprocessing:**
 - Operations like resizing, normalization, and format conversion are applied to standardize the images for compression.

3. Compression Layer:

- **LZW Compression Algorithm:**
 - The core of the system, where the LZW (Lempel-Ziv-Welch) algorithm is applied to compress the medical images.
 - The algorithm identifies repeated patterns in the image data, replaces them with shorter codes, and compresses the image without losing any critical information.

4. Storage Layer:

- **Compressed Image Storage:**
 - The compressed images are stored in a database or cloud storage system for easy retrieval.
 - The storage system must ensure security and compliance with healthcare data regulations (e.g., HIPAA).

5. Transmission Layer:

- **Image Transmission:**

- The compressed images are transmitted over the network to healthcare providers, specialists, or telemedicine platforms.
- The transmission process must be optimized for bandwidth efficiency and data security.

6. Decompression Layer:

- **LZW Decompression Algorithm:**

- On the receiving end, the compressed images are decompressed using the LZW algorithm to restore them to their original quality for diagnosis and analysis.

7. Application Layer:

- **Telemedicine Integration:**

- The decompressed images are integrated into telemedicine platforms where healthcare providers can analyze and diagnose based on the images.
- The system could also include features like image viewing, annotations, and patient data management.

8. User Interface Layer:

- **Healthcare Providers' Interface:**

- A user-friendly interface allows doctors and specialists to upload, view, and download medical images.
- The interface includes tools for zooming, rotating, and analyzing the images.

9. Security Layer:

- **Data Encryption and Compliance:**

- Security protocols are implemented to protect patient data during storage and transmission.

- The system must comply with healthcare regulations to ensure patient confidentiality.

4.2 UML DIAGRAMS

Use Case Diagram

The use case diagram for LZW Driven Medical Image Compression system illustrates the interactions between the system and its users. In this diagram, the primary actors include User and Software. For a user, use cases involve uploading of image. The software then checks if the image is grayscale or not. After identifying the image as grayscale the software then applies LZW algorithm to compress the image. The software then encrypts the compressed image and returns to the user. The user then sends the encrypted file with original password. The receiver decrypts to get the compressed image.

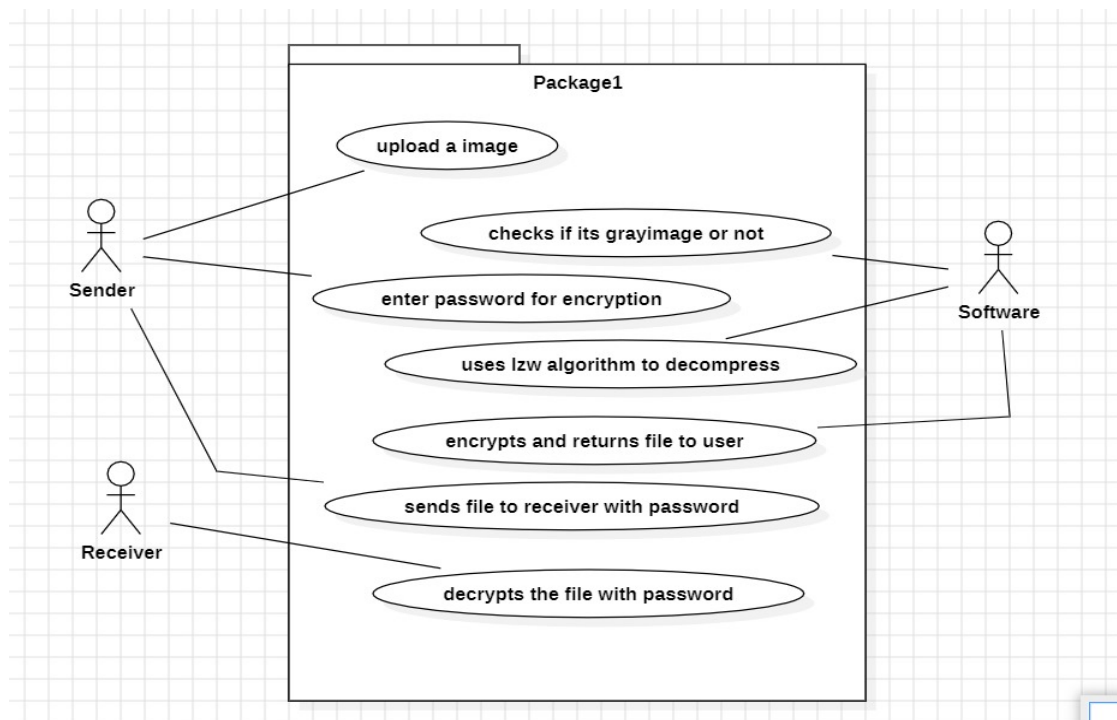


Fig 4.2.1 Usecase Diagram

Sequence Diagram

The sequence diagram for LZW Driven Medical Image Compression outlines the sequence of interactions between users and software. For instance, the user starts by first uploading a image. The software then checks if its grayscale or not. If grayscale then applies LZW algorithm to compress the image and encrypts the image with password provided by user. The user then sends the encrypted file to another user .The receiver decrypts and retrieves the compressed image using original password.

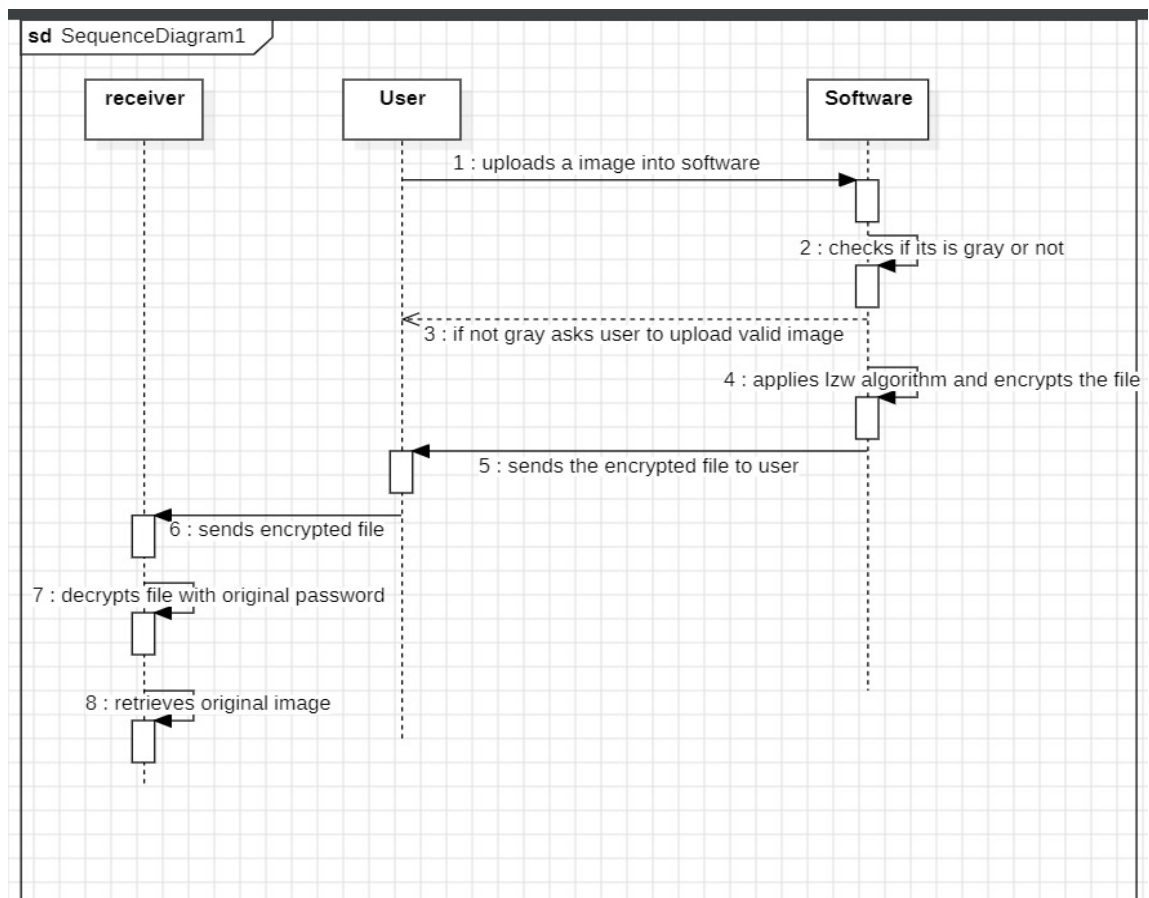


Fig 4.2.2 Sequence Diagram

Activity Diagram

The activity diagram for LZW Driven Medical Image Compression represents the workflow of specific processes within the system. For example, the process of compressing a image involves several steps such as uploading image, verifying the image, applying LZW algorithm, encrypting the compressed file, decrypting the file at the end of the receiver using the original password. This helps in visualizing the dynamic aspects of the system and understanding how different activities are coordinated.

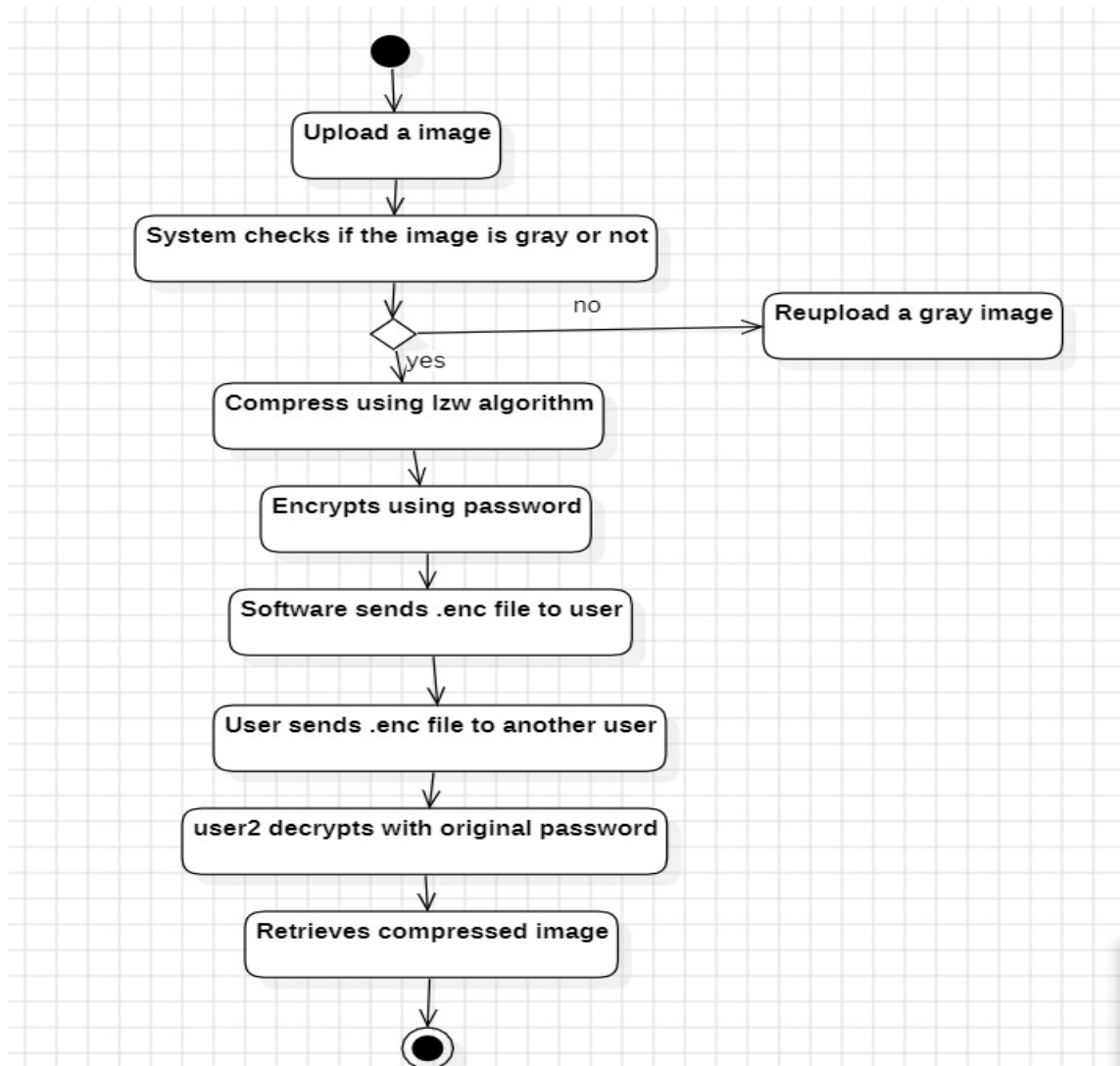


Fig 4.2.3 Activity Diagram

5. IMPLEMENTATION

1. Setup and Requirements:

- **Development Environment:**
 - Install necessary software, including Python or C++ IDE (e.g., PyCharm, Visual Studio Code).
 - Install libraries such as OpenCV for image processing, NumPy for handling arrays, and Matplotlib for visualizing images.
- **Hardware Setup:**
 - Ensure that the system has sufficient processing power and memory to handle large medical images.
- **Data Collection:**
 - Obtain sample medical images (MRI, CT, X-ray) for testing, either from open datasets or hospital collaborations.

2. Image Preprocessing:

- **Image Loading:**
 - Write scripts to load medical images into the system.
- **Preprocessing:**
 - Perform operations such as resizing, grayscale conversion, and normalization to standardize image formats.

3. LZW Compression Implementation:

- **Algorithm Development:**
 - Implement the LZW algorithm by writing code that reads image data, identifies repeated patterns, and replaces them with codes.
 - Ensure that the compression process is lossless, preserving image quality.
- **Compression Function:**
 - Develop a function to apply the LZW algorithm to the image data,

reducing the file size while maintaining image integrity.

- **Testing Compression:**

- Test the compression function on various medical images and evaluate the compression ratio and image quality.

4. Storage and Retrieval System:

- **Database or Cloud Storage Setup:**

- Set up a local database or cloud storage system (e.g., AWS S3, Google Cloud Storage) to store compressed images securely.

- **Storage Functionality:**

- Write scripts to save and retrieve compressed images from storage.

5. Image Transmission:

- **Transmission Protocol:**

- Implement secure transmission protocols (e.g., HTTPS, SSL/TLS) to send compressed images over the network.

- **Bandwidth Optimization:**

- Optimize the system to handle low-bandwidth scenarios, ensuring fast and reliable image transmission.

6. Decompression Implementation:

- **LZW Decompression Algorithm:**

- Write the decompression algorithm to reverse the LZW compression process and restore images to their original form.

- **Testing Decompression:**

- Test the decompression function with compressed images and compare the output with the original images to ensure accuracy.

7. Telemedicine Integration:

- **User Interface Development:**

- Create a user-friendly interface for healthcare providers to upload, view, and download images.
- Integrate image viewing tools (e.g., zoom, rotate) and patient data management features.

- **System Integration:**

- Integrate the image compression and transmission system with existing telemedicine platforms.

8. Security and Compliance:

- **Data Encryption:**

- Implement encryption methods to secure patient data during storage and transmission.

- **Compliance Checks:**

- Ensure that the system complies with healthcare regulations such as HIPAA to protect patient confidentiality.

5.2 SAMPLE CODE

1.app.py

```
from flask import Flask, render_template, request, redirect, url_for, send_file
import numpy as np
import cv2
import os
import subprocess
app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
DECOMPRESSED_FOLDER = 'decompressed'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)
if not os.path.exists(DECOMPRESSED_FOLDER):
    os.makedirs(DECOMPRESSED_FOLDER)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['DECOMPRESSED_FOLDER'] = DECOMPRESSED_FOLDER

def compress(image):
    flattened_image = image.flatten()
    dictionary_size = 256
    dictionary = {chr(i): i for i in range(dictionary_size)}
    code_word = ""
    compressed_data = []
    for pixel_value in flattened_image:
        new_code_word = code_word + chr(pixel_value)
        if new_code_word in dictionary:
            code_word = new_code_word
        else:
            compressed_data.append(dictionary[code_word])
            dictionary[new_code_word] = dictionary_size
            dictionary_size += 1
            code_word = chr(pixel_value)

    compressed_data.append(dictionary[code_word])
    original_size = len(flattened_image) * 32
    compressed_size = len(compressed_data) * 32 # Assuming 32-bit for each
    compression_ratio = original_size / compressed_size
    return compressed_data, dictionary, compression_ratio

def decompress(compressed_data):
    dictionary_size = 256
    inverse_dictionary = {i: chr(i) for i in range(dictionary_size)}
    code_word = chr(compressed_data.pop(0))
    decompressed_data = [code_word]
    for code in compressed_data:
        if code in inverse_dictionary:
```

```

        entry = inverse_dictionary[code]
    elif code == dictionary_size:
        entry = code_word + code_word[0]
    else:
        raise ValueError('Bad compression')
    decompressed_data.append(entry)
    inverse_dictionary[dictionary_size] = code_word + entry[0]
    dictionary_size += 1
    code_word = entry
    return ".join(decompressed_data)

def encrypt_file(input_filepath, output_filepath, password):
    command = f"openssl enc -aes-256-cbc -salt -pbkdf2 -in {input_filepath} -out {output_filepath} -k {password}"
    subprocess.run(command, shell=True)

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == "":
            return redirect(request.url)
        if file:
            password = request.form['password']
            if not password:
                return "Password is required.", 400
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(filepath)
            image = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)
            compressed_data, dictionary, compression_ratio = compress(image)
            decompressed_data = decompress(compressed_data)
            decompressed_int_values = [ord(char) for char in decompressed_data]
            if len(decompressed_int_values) != np.prod(image.shape):
                raise ValueError("Decompressed data length does not match the original image.")
            decompressed_image = np.array(decompressed_int_values, dtype=np.uint8).reshape(image.shape)
            decompressed_filepath = os.path.join(app.config['DECOMPRESSED_FOLDER'], 'decompressed_' + file.filename)
            cv2.imwrite(decompressed_filepath, decompressed_image)

            # Encrypt the decompressed image
            encrypted_filepath = decompressed_filepath + '.enc'
            encrypt_file(decompressed_filepath, encrypted_filepath, password)

    return render_template('result.html',
                           original=file.filename,

```

```

        decompressed='decompressed_' + file.filename,
        encrypted='decompressed_' + file.filename + '.enc',
        original_size=image.size * image.itemsize,
compressed_size=len(compressed_data),          # Assuming 32-bit for
each entry                                     compression_ratio=compression_ratio)

    return render_template('upload.html')

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_file(os.path.join(app.config['UPLOAD_FOLDER'], filename))

@app.route('/decompressed/<filename>')
def decompressed_file(filename):
    return send_file(os.path.join(app.config['DECOMPRESSED_FOLDER'], filename))

@app.route('/encrypted/<filename>')
def encrypted_file(filename):
    return send_file(os.path.join(app.config['DECOMPRESSED_FOLDER'], filename))

if __name__ == '__main__':
    app.run(debug=True)

```

2.decrypt.html

```

<!doctype html>
<html>
<head>
<title>Decrypt File</title>
</head>
<body>
<h1>Decrypt File</h1>
<form method="post" enctype="multipart/form-data">
<input type="file" name="file">
<input type="text" name="password" placeholder="Enter password">
<input type="submit" value="Decrypt">
</form>
</body>
</html>

```

3.result.html

```

<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Compression Result</title>
</head>

```

```

<body>
  <h1>File Processed</h1>
  <p>Original Image: <a href="{{ url_for('uploaded_file', filename=original) }}">{{
original }}</a></p>
  <p>Decompressed Image: <a href="{{ url_for('decompressed_file',
filename=decompressed) }}">{{ decompressed }}</a></p>
  <p>Encrypted Image: <a href="{{ url_for('encrypted_file', filename=encrypted)
 }}">{{ encrypted }}</a></p>
  <p>Original Size: {{ original_size }} bytes</p>
  <p>Compressed Size: {{ compressed_size }} bytes</p>
  <p>Compression Ratio: {{ compression_ratio }}</p>

  <h2>Decrypting the Image</h2>
  <p>To decrypt the image on your device, use the following command:</p>
  <pre>
openssl enc -aes-256-cbc -d -pbkdf2 -in {{ encrypted }} -out decrypted_{{ original }}
-k [YOUR_PASSWORD]
  </pre>
  <p>Replace [YOUR_PASSWORD] with the password you provided during
upload.</p>
</body>
</html>

```

4. upload.html

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Upload File</title>
  </head>
  <body>
    <h1>Upload File</h1>
    <form method="post" enctype="multipart/form-data">
      <input type="file" name="file">
      <br>
      <label for="password">Encryption Password:</label>
      <input type="password" name="password" id="password">
      <br>
      <input type="submit" value="Upload">
    </form>
  </body></html>

```

6. TESTING

1.Functional Testing

➤ Test Case 1: Image Compression

- Objective: Verify that the LZW algorithm compresses an image correctly.
- Input: A sample medical image (e.g., MRI or CT scan).
- Expected Result: The image size should be reduced after compression without losing quality.

➤ Test Case 2: Image Decompression

- Objective: Ensure that the compressed image can be decompressed correctly to its original form.
- Input: Compressed medical image.
- Expected Result: The decompressed image should match the original image in quality and content.

➤ Test Case 3: Compression Ratio

- Objective: Check that the algorithm achieves the expected compression ratio.
- Input: Multiple medical images of different sizes.
- Expected Result: The compression ratio should align with the project's goal, indicating efficient compression.

2.Performance Testing

➤ Test Case 4: Compression Speed

- Objective: Measure the time taken to compress different sizes of images.
- Input: Small, medium, and large medical images.
- Expected Result: Compression should be completed within an acceptable time frame.

➤ Test Case 5: Decompression Speed

- Objective: Evaluate the time taken to decompress images.
- Input: Compressed images.
- Expected Result: Decompression should be quick and efficient.

3. Usability Testing

➤ Test Case 6: User Interface

- Objective: Test the interface through which users can compress and decompress images.
- Input: A user trying to upload, compress, and download images.
- Expected Result: The interface should be user-friendly and allow seamless operation.

4. Compatibility Testing

➤ Test Case 7: Image Format Support

- Objective: Verify that the system supports various medical image formats (e.g., DICOM).
- Input: Images in different formats.
- Expected Result: The system should be able to compress and decompress images in various formats without issues.

5. Stress Testing

➤ Test Case 8: Large Data Handling

- Objective: Test the system's capability to handle a large number of images or very large image files.
- Input: Bulk upload of large medical images.
- Expected Result: The system should compress and decompress the images without crashing or significant delay.

6. Edge Case Testing

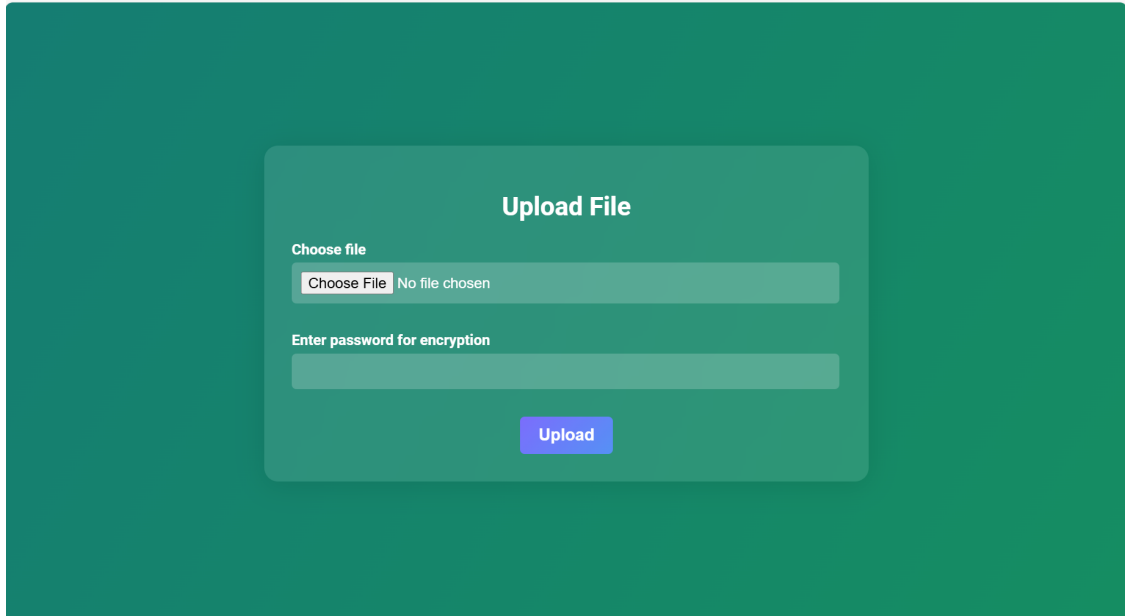
➤ Test Case 9: Empty Image Input

- Objective: Check how the system handles an empty image or zero-byte file.
- Input: An empty file.
- Expected Result: The system should return an appropriate error message without crashing.

➤ Test Case 10: Corrupted Image Input

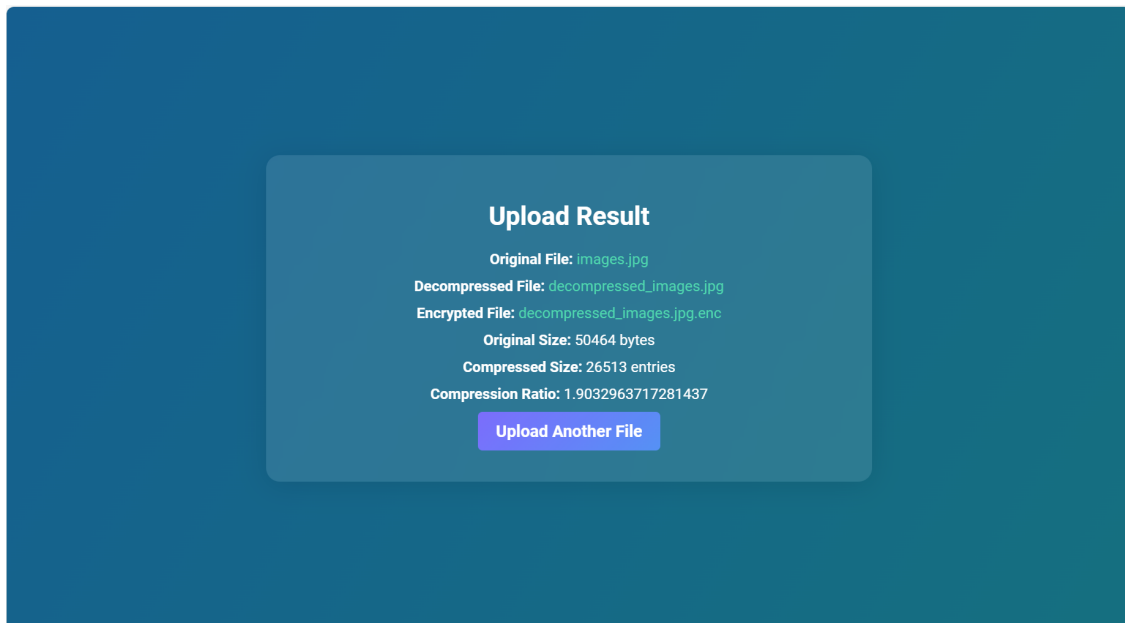
- Objective: Test how the system deals with corrupted image files.
- Input: A corrupted medical image.
- Expected Result: The system should detect the issue and provide feedback.

7. OUTPUT SCREENS



The screenshot shows a web interface with a dark green background. In the center, there is a lighter green rounded rectangle containing the 'Upload File' form. The form has a title 'Upload File' at the top. Below it, there is a section 'Choose file' with a button labeled 'Choose File' and the text 'No file chosen'. Below that, there is a section 'Enter password for encryption' with a text input field. At the bottom of the form is a blue button labeled 'Upload'.

Fig 7.1 Upload File Page



The screenshot shows a web interface with a dark blue background. In the center, there is a lighter blue rounded rectangle containing the 'Upload Result' information. The form has a title 'Upload Result' at the top. Below it, there is a list of file details: 'Original File: images.jpg', 'Decompressed File: decompressed_images.jpg', 'Encrypted File: decompressed_images.jpg.enc', 'Original Size: 50464 bytes', 'Compressed Size: 26513 entries', and 'Compression Ratio: 1.9032963717281437'. At the bottom of the form is a blue button labeled 'Upload Another File'.

Fig 7.2 Uploaded Result Page

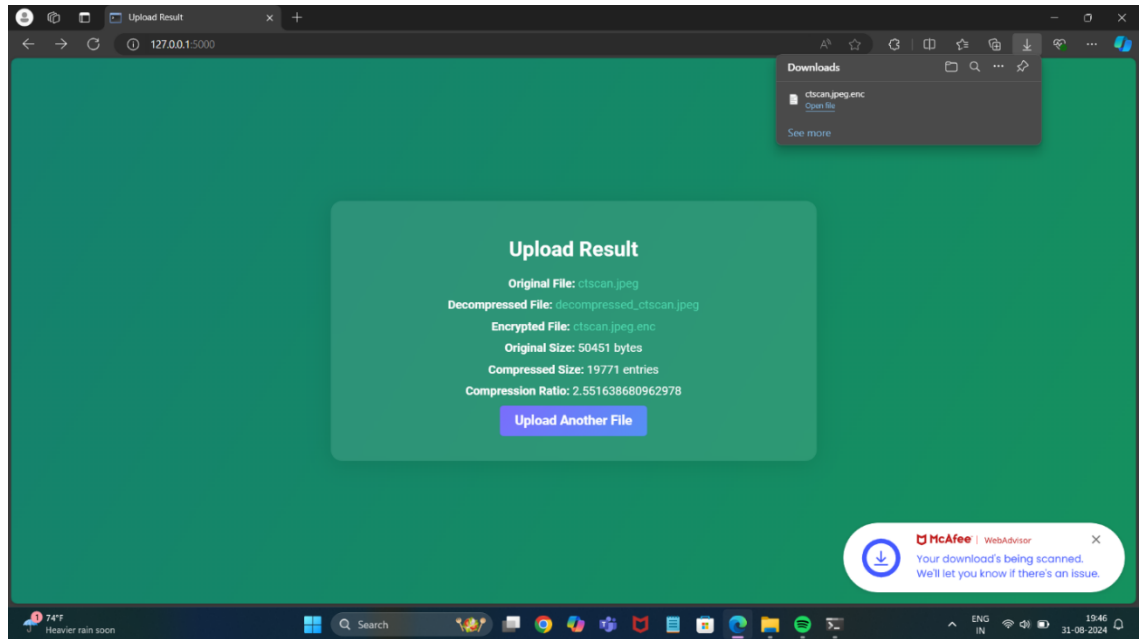


Fig 7.3 Decompressed Download Page

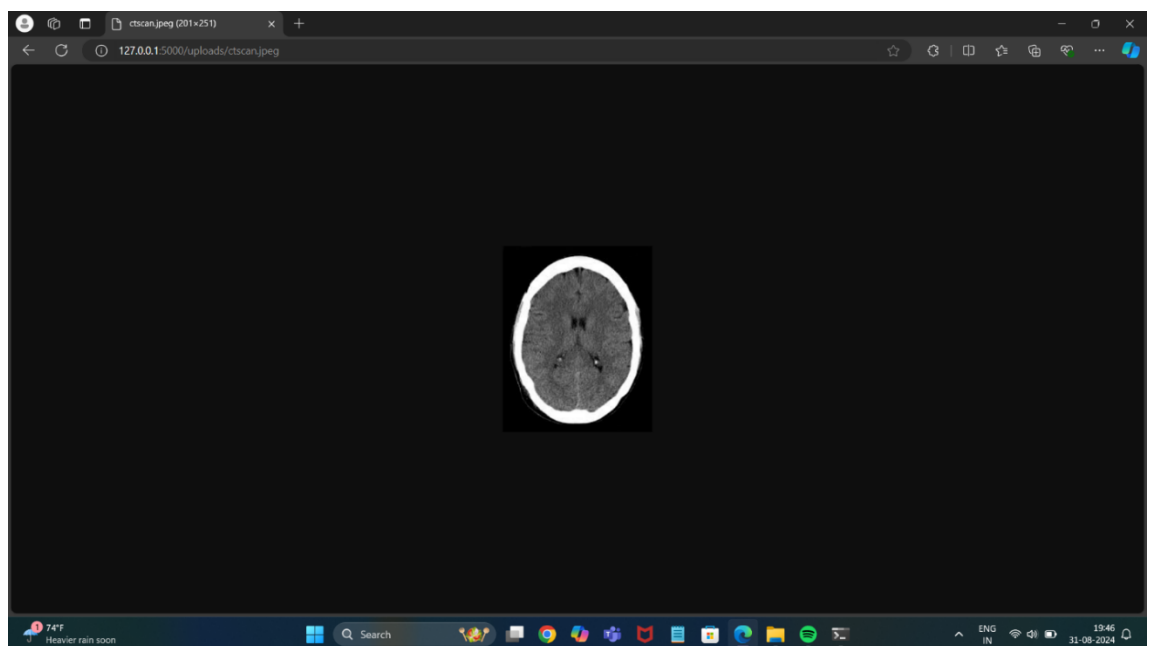


Fig 7.4 Original Image

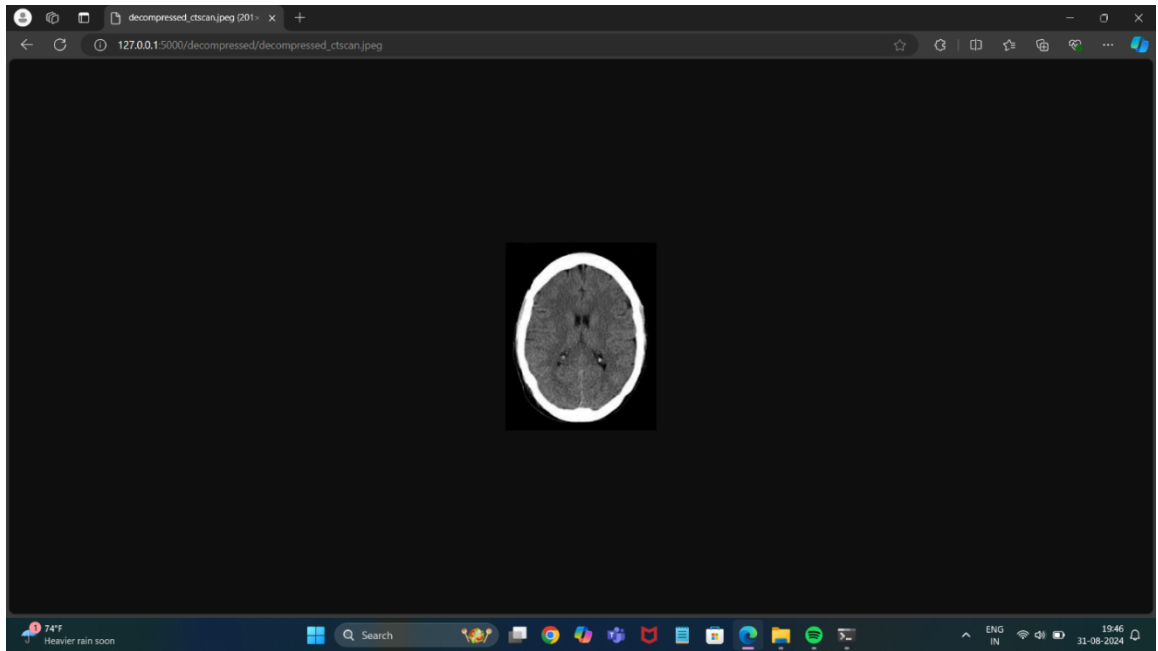


Fig 7.5 Decompressed Image

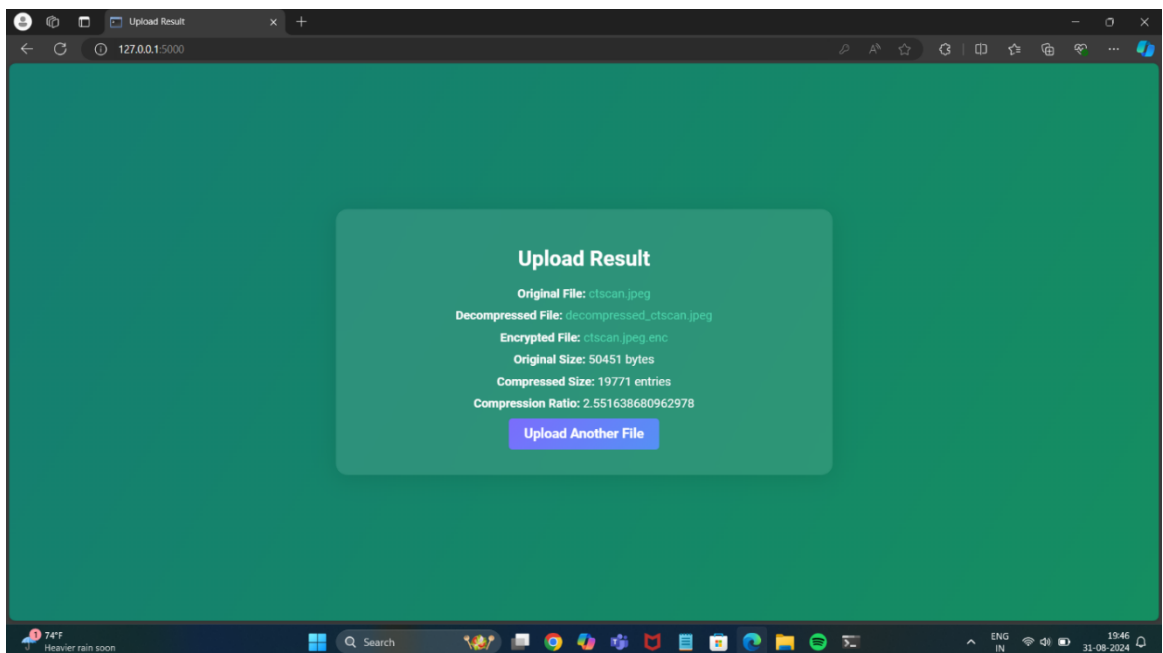


Fig 7.6

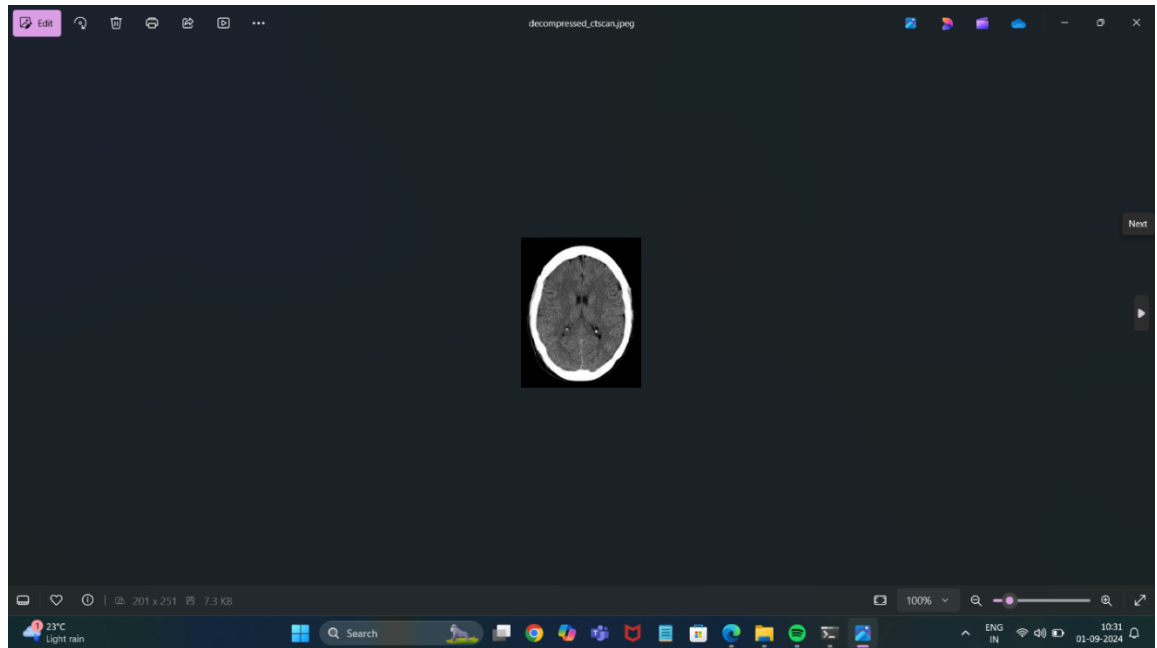


Fig 7.7 Downloaded Image

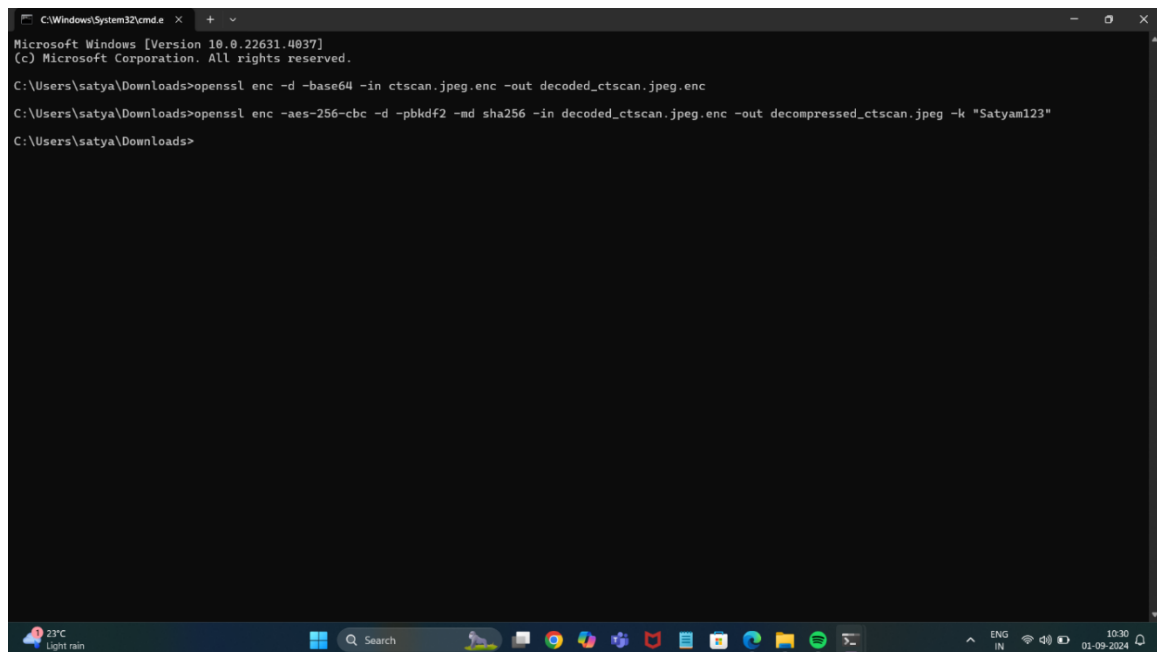


Fig 7.8 Encryption and Decryption command

8. CONCLUSION

8.1 CONCLUSION

The application of LZW (Lempel-Ziv-Welch) compression in medical imaging offers a powerful solution for reducing file sizes while preserving image integrity and diagnostic value. Through lossless compression, LZW ensures that critical medical data remains unchanged, enabling accurate diagnostics and patient care. This study demonstrates significant reductions in storage requirements and transmission times for medical images without compromising quality. As medical data grows exponentially, implementing efficient compression algorithms like LZW is crucial for optimizing healthcare storage and network resources. Future research directions include combining LZW with other techniques to enhance compression efficiency and exploring its adaptability to various medical imaging modalities.

8.2 FURTHER ENHANCEMENTS

1. Visualization: Include images or graphs to illustrate the compression process, such as:

- Original vs. compressed images
- Compression ratio vs. image quality
- Histograms of pixel values before and after compression

2. Quantitative analysis: Provide more detailed numerical results, such as:

- Compression ratios for different image modalities (e.g., MRI, CT, X-ray)
- Peak signal-to-noise ratio (PSNR) or other quality metrics
- Computational time and memory usage for compression and decompression

3. Comparative study: Compare LZW with other compression algorithms, such as:

- Huffman coding
- Arithmetic coding
- JPEG-LS
- Wavelet-based compression

4. Clinical relevance: Discuss the clinical implications of using LZW compression in medical imaging, such as:

- Impact on diagnostic accuracy
- Effects on image interpretation and patient care
- Potential applications in telemedicine or remote healthcare

5. Future directions: Explore potential avenues for further research, such as:

- Combining LZW with other compression techniques
- Adaptive compression based on image content or quality requirements
- Hardware acceleration or parallel processing for faster compression

9. BIBLIOGRAPHY

9.1 REFERENCES

1. GeeksforGeeks. (2024) provides a detailed overview of the LZW (Lempel-Ziv-Welch) compression technique, explaining the steps involved in compressing data efficiently using this algorithm.
2. Wikipedia. (2024) presents an in-depth look into the history, development, and technical aspects of the LZW compression method, including its origin as part of the Lempel-Ziv family of algorithms.
3. TechTarget. (2024), in their article, describes the practical applications and benefits of the LZW compression technique, focusing on its use in image compression and its efficiency compared to other methods.

10. APPENDICES

Appendix A: Project Setup

1. Dependencies:

○ Python Libraries:

- OpenCV
- NumPy
- Cryptography
- smtplib (part of Python's standard library)
- Email handling (part of Python's standard library)

- **Installation Instructions:** Instructions for installing the necessary libraries.

2. Environment Configuration:

- Steps to set up a Python environment (e.g., using `venv` or `conda`).

Appendix B: LZW Compression Class

1. Class Overview:

- Description of the LZW compression algorithm and how it is used in the project for data compression and decompression.
- Explanation of the initialization, compression, and decompression processes.

2. Usage Instructions:

- How to create an instance of the LZW class.
- How to use the class methods to compress and decompress data.

Appendix C: Image Encryption

1. Encryption Process:

- Overview of the encryption process using the `openssl` algorithm.
- How data is converted into a secure encrypted format.

2. Key Management:

- How to generate and manage encryption keys.

- Guidelines for storing and sharing keys securely.

Appendix D: File Formats and Data Handling

1. File Formats:

- Description of file formats used, such as .jpg for images and .dat for encrypted data.

2. Data Conversion:

- Overview of how data is converted between different formats (e.g., image to list, list to string, string to bytes).

Appendix E: Example Outputs

1. Sample Images:

- Description of the types of images used in the project and how they are processed.
- Explanation of the results after image decryption.

2. Sample Encrypted Data:

- Overview of the encrypted data format and how it is stored.

Appendix F: Security Considerations

1. Handling Sensitive Information:

- Best practices for managing and sharing encryption keys.
- Recommendations for handling email credentials securely.

2. Encryption Best Practices:

- General recommendations for encrypting and decrypting data effectively.