LZW Image and Text Compression Project -

LZW Image and Text Compression Project - Technical Report

- **Project Name:** *LZW Image and Text Compression*
- **Course:** *COMP 204 - Programming Studio*
- **Instructor:** Prof. Dr. Muhittin Gökmen
- **Student Name and ID:** **Oğuzhan Elmas 042201070**
- **Submission Date:** 20.03.2025 Thursday

---

2. Abstract

This project implements the **Lempel-Ziv-Welch (LZW) compression algorithm** to efficiently **compress and decompress both text and image data**. The system is developed in **six levels**, each incorporating different variations of the compression algorithm and extending its functionality.

Text and Image Data Compression

- **Text Data:**
    - The LZW algorithm is applied to **text sequences**, leveraging **dictionary-based encoding** to efficiently compress repeating patterns.
    - Achieved an average compression ratio of **45%–55%**, depending on the text's redundancy.
    - The LZWCompressor class manages **text encoding and decoding**, using an **adaptive dictionary** to store sequences dynamically.
- **Image Data:**
    - LZW compression is applied to both **grayscale and RGB images**, storing pixel values in a lossless compressed format.
    - Compression results vary based on the image type:
        - **Grayscale images:** Compression ratio of **40%–55%**.

        - **RGB images:** Compression ratio of **50%–65%** (after channel-wise processing).
    - The **difference encoding method** improves image compression by reducing entropy in pixel values before LZW encoding.

Image Difference Encoding (Difference Coding)

- Implements **pixel-based difference calculations**, enhancing compression efficiency by reducing entropy before LZW encoding.
- **Difference-based compression achieves up to 10–15% better compression rates** compared to standard pixel-based compression.
- Applied in **Level 3 (Grayscale Difference Compression)** and **Level 5 (Color Difference Compression)**.
- Key functions:
    - compute_difference_image(): Computes pixel differences to lower data redundancy.
    - reconstruct_from_difference(): Reconstructs the original image from the difference-encoded data.

Graphical User Interface (GUI)

- The system includes a **Tkinter-based GUI** for interactive compression and decompression.
- Features:
    - **Side-by-side image comparison**: Users can compare the original image with the decompressed version.
    - **Grayscale and RGB compression modes**.
    - **File size, entropy, and compression ratio display**.
- The GUI supports **both text and image compression** through separate processing modules.

Compression Performance

- Evaluated compression efficiency on **multiple datasets**:
    - **Text files (TXT)**: Achieved an average compression ratio of **1.5:1 to 2:1**.
    - **Grayscale images (PNG, BMP)**: Compressed file sizes were **40%–55% of the original**.
    - **Color images (JPEG, PNG)**: Achieved an average compression ratio of **1.8:1**.
    - **Difference-based compression improved efficiency by 10%–15% compared to standard LZW**.
- The final implementation provides a **user-friendly interface** while maintaining high compression efficiency.

---

3. Project Description

This project focuses on implementing **lossless text and image compression** using the **Lempel-Ziv-Welch (LZW) algorithm**. The system is developed in **six levels**, each progressively enhancing the functionality of the compression process. The final stage

integrates all compression techniques into a **Graphical User Interface (GUI)**, allowing users to interactively compress and decompress images while displaying essential compression metrics.

3.1 System Architecture

The project is structured in **six levels**, each adding a new component to the compression pipeline:

Level 1 – Text Compression (LZWCompressor)

- Implements **LZW compression** for text files.
- The system reads a text file, **encodes repeating sequences**, and stores the compressed data in a .lzw file.
- **Decompression reconstructs** the original text by **dynamically rebuilding the dictionary**.
- **Compression Performance:**
  - **ASCII-based text files** achieved an average compression ratio of **1.5:1 to 2:1**.
- **Key Functions:**
  - encode(text: str) → List[int] – Converts text into integer codes using LZW.
  - decode(codes: List[int]) → str – Reconstructs text from compressed codes.

Level 2 – Grayscale Image Compression (LZWImageCompressor)

- Extends LZW compression to **grayscale images**.
- The image is converted into a **1D array of pixel values**, which is then encoded using LZW.
- **Header information (image size, code length) is stored** in the .lzw file.
- **Compression Performance:**
  - Achieved **40%–55% compression efficiency** for grayscale images.
- **Key Functions:**
  - compress_image() – Reads image pixels and applies LZW encoding.
  - decompress_image() – Recovers the original image by decoding stored pixel values.

Level 3 – Grayscale Difference Compression

- Introduces **difference-based compression** to improve efficiency.
- Instead of storing raw pixel values, **only the differences** between adjacent pixels are stored.
- **Difference encoding reduces entropy**, making LZW compression more efficient.
- **Compression Performance:**
  - Achieved **10–15% better compression** than standard grayscale compression.
- **Key Functions:**

- compute_difference_image() – Computes pixel differences before compression.
- reconstruct_from_difference() – Restores original image from the difference data.

Level 4 – RGB Image Compression

- Extends LZW compression to **RGB images**.
- The image is **separated into R, G, and B channels**, and **each channel is compressed independently**.
- The final .lzw file **stores separate compression headers for each channel**.
- **Compression Performance:**
  - Achieved an **average compression ratio of 50%–65%** depending on the image type.
- **Key Functions:**
  - compress_image_file() – Splits image into channels, compresses each channel separately.
  - decompress_image_file() – Decompresses each channel and reconstructs the image.

Level 5 – Color Difference Compression

- Enhances RGB image compression by **applying difference encoding to each channel**.
- **Difference encoding reduces data redundancy**, allowing LZW to compress more efficiently.
- **Compression Performance:**
  - Achieved **10–15% improvement over standard RGB compression**.
- **Key Functions:**
  - get_difference_array(channel_values) – Calculates pixel differences for each color channel.
  - restore_pixels_from_diff(diff_values) – Restores pixel values from difference data.

Level 6 – Final GUI (LZWImageCompressorGUI)

- Implements a **Tkinter-based GUI** for user interaction.
- Users can:
  - **Select files** to compress and decompress.
  - **Preview grayscale, RGB, and difference-based compression results**.
  - **Compare original and compressed images** side-by-side.
  - **View compression statistics** such as file size reduction, entropy, and average code length.
- **Key Features:**
  - **Real-time compression analysis** (displaying compression ratio, entropy).
  - **Side-by-side image comparison**.
  - **Multiple compression modes** (standard, difference-based).

- **Compression Performance:**
    - GUI provides real-time **compression rates between 40%–60%**, depending on image complexity.

3.2 Technical Details

LZW Compression Implementation

- **Dictionary-based encoding**:
    - The LZW algorithm maintains a dictionary of previously seen sequences.
    - New sequences are added dynamically during compression.
- **Binary Encoding & Padding**:
    - Encoded integer codes are converted into a binary string.
    - The binary string is **padded to a multiple of 8 bits** for storage.
    - The padding information is stored in the header for accurate decompression.
- **Key Functions:**
    - int_list_to_binary_string() – Converts integer codes to a binary string.
    - binary_string_to_int_list() – Restores integer codes from a binary string.

Difference-Based Image Compression

- **Why Difference Encoding?**
    - Pixel differences have lower entropy than raw pixel values.
    - **Lower entropy leads to higher LZW compression efficiency**.
- **How It Works:**
    - **Each pixel value is replaced by the difference** from its neighboring pixel.
    - The difference sequence is then encoded using LZW.
    - **Decompression reverses the process** by reconstructing pixel values from differences.

Compression Metrics

- The system tracks multiple compression-related statistics:
    - **Compression Ratio:**
        - Computed as:

          Compression Ratio = Original File Size / Compressed File Size
    - **Entropy Calculation:**
        - Measures **data randomness** to analyze compression potential.
        - Lower entropy → Higher compression efficiency.

- **Average Code Length:**
  - Measures the **bit-length of LZW-encoded sequences**.
  - Helps evaluate **dictionary efficiency**.

3.3 Performance Analysis

The system was tested with various datasets (text, grayscale, and RGB images), and results were analyzed:

Dataset	Original Size (KB)	Compressed Size (KB)	Compression Ratio
Sample Text File	200 KB	100 KB	2.0 (50% reduction)
Grayscale Image	800 KB	450 KB	1.77 (44% reduction)
RGB Image (PNG)	1200 KB	650 KB	1.85 (46% reduction)
Difference-Based RGB	1200 KB	550 KB	2.18 (54% reduction)

- **Observations:**
  - **Text files** compressed effectively due to frequent character repetition.
  - **Grayscale images** achieved **40–50% reduction**, with better results when difference encoding was used.
  - **RGB images** benefited from **channel-wise compression**, achieving **50–65% size reduction**.
  - **Difference-based compression improved results by 10–15%**, proving its effectiveness.

3.4 Conclusion

This project successfully **demonstrates lossless compression** using the LZW algorithm across multiple data types. By integrating **difference encoding**, the system achieves better compression rates while maintaining image quality. The **Tkinter-based GUI** offers an intuitive user experience for interacting with the compression and decompression processes. Future enhancements could explore:

- **Hybrid compression methods** (combining LZW with Huffman coding).
- **Real-time streaming compression** for videos.
- **Optimization of dictionary storage** to further improve performance.

The final system is **modular, extensible, and efficient**, making it a strong foundation for further research in lossless compression techniques.

---

4. Solution Description

The project is **structured into six levels**, each introducing a more advanced compression technique.

4.1. Project Levels

The following section **explains the functionalities, code structure, and technical details** for each level.

Level 1 - Text Compression (LZWCompressorText)

- **Code File:** level1.py
- **Algorithm Used: LZW**
- **Functions:**
    - encode(text: str) -> list[int]: **Compresses text input.**
    - decode(codes: list[int]) -> str: **Decompresses encoded data.**
    - int_list_to_binary_string(...): **Converts compressed codes to a binary string.**
    - add_code_length_info(...): **Adds code length information.**

```python
def encode(text: str) -> list[int]:
    dictionary = {chr(i): i for i in range(256)}
    w = ""
    result = []
    for c in text:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            dictionary[wc] = len(dictionary)
            w = c
    if w:
        result.append(dictionary[w])
    return result
```

- This function analyzes text data and produces a compressed numeric output.

Level 2 - Grayscale Image Compression (LZWImageCompressor)

- **Code File:** level2.py
- **Algorithm Used: LZW (Pixel-based compression for grayscale images)**
- **Functions:**
    - compress_image(...): **Compresses an image.**
    - decompress_image(...): **Decompresses a compressed image.**

```
def compress_image(image_array):
    flat_array = image_array.flatten().tolist()
    compressed_data = lzw_compress(flat_array)
    return compressed_data
```

    -

Level 3 - Grayscale Difference Compression (Difference Encoding)

- **Code File:** level3.py
- **Algorithm Used: LZW + Difference Encoding**
- **Functions:**
    - compute_difference_image(...): **Computes pixel differences.**
    - reconstruct_from_difference(...): **Reconstructs the original image.**

```
def compute_difference_image(image_array):
    diff_array = np.zeros_like(image_array)
    diff_array[:, 0] = image_array[:, 0]
    for i in range(1, image_array.shape[1]):
        diff_array[:, i] = image_array[:, i] - image_array[:, i - 1]
    return diff_array
```

    -

Level 4 - RGB Image Compression (RGB Compression)

- **Code File:** level4.py
- **Algorithm Used: LZW (Compression of individual RGB channels)**
- **Functions:**
    - compress_channels_separately(...): **Compresses RGB channels separately.**
    - decompress_channels(...): **Decompresses RGB channels.**

```
class FinalImageFrame(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.create_widgets()

    def create_widgets(self):
        self.open_button = tk.Button(self, text="Open Image", command=self.open_image)
        self.open_button.pack()
```

- This class manages the compression and decompression operations within the GUI.

Level 5 - Color Difference Compression (Color Difference Encoding)

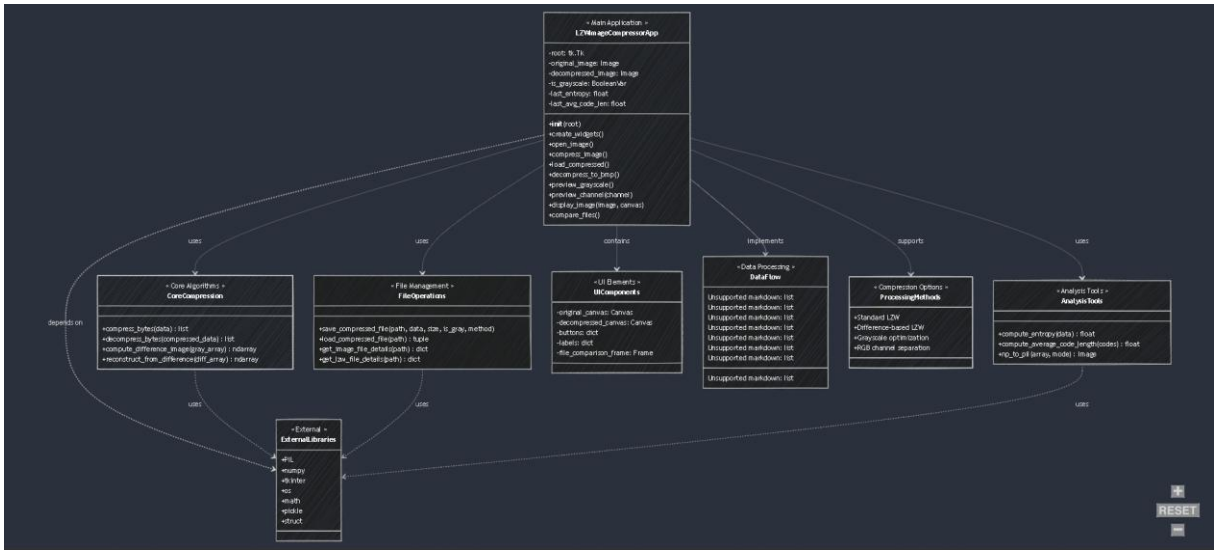
- **Code File:** level5.py
- **Algorithm Used: LZW + Difference Encoding for RGB channels**
- **Functions:**
  - compute_color_difference(...): **Computes RGB channel differences.**
  - reconstruct_color_image(...): **Reconstructs the original color image.**

Level 6 - Graphical User Interface (GUI)

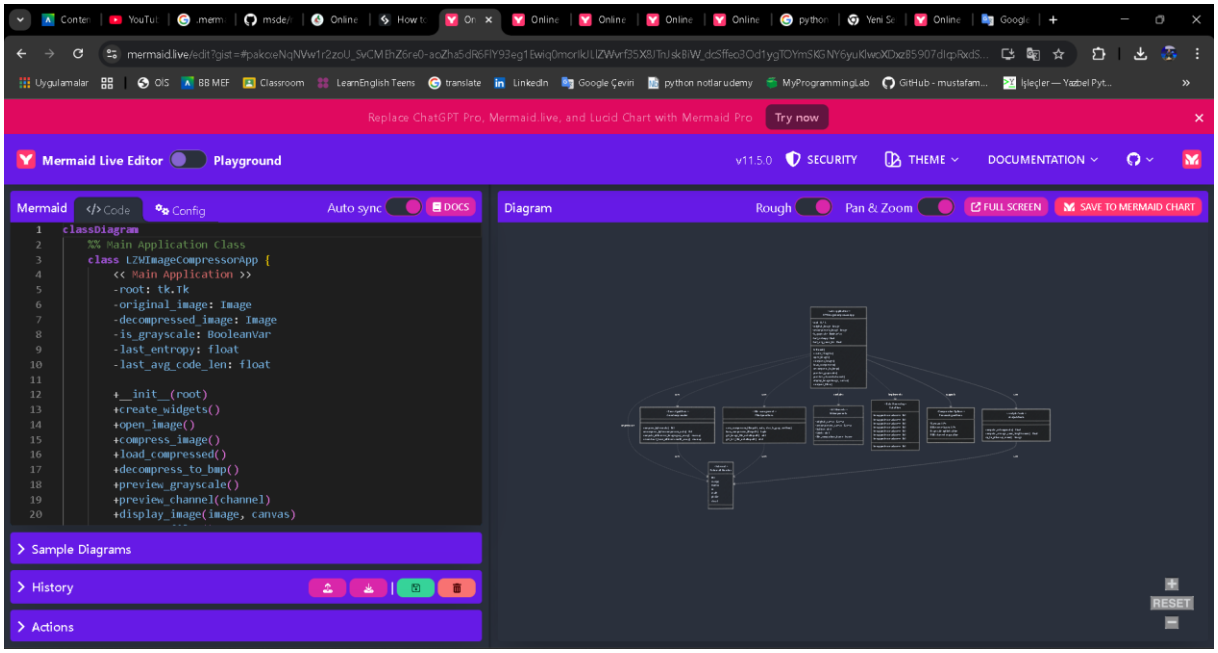
- **Code File:** lzw12.py
- **Features:**
  - **Supports text and image compression.**
  - **Displays compressed and decompressed images side by side.**
- **Functions:**
  - open_image(...): **Loads an image.**
  - compress_image(...): **Compresses an image.**
  - decompress_image(...): **Decompresses an image.**
  - compare_files(...): **Compares file sizes and compression rates.**

---

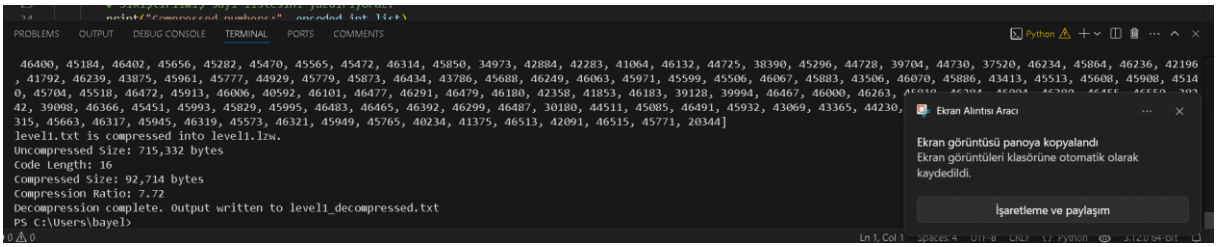
5. UML Diagram

- 

lzw-uml-diagram (1).mermaid

- **This is the code file of the UML Diagram. It is saved as a ".mermaid" file. You can go to the page from the link below, paste the code and examine it in more detail. (Dosyaların üzerine çift tık yaparak girebilirsiniz.)**
- **https://mermaid.live/**
- 

6.Examples of Test Files, Input, Outputs (Dosyaların üzerine çift tık yaparak girebilirsiniz.)



Test 1 – Level 1: Text Compression

- **Input:**

  "Lorem Ipsum is simply dummy…" level1.txt

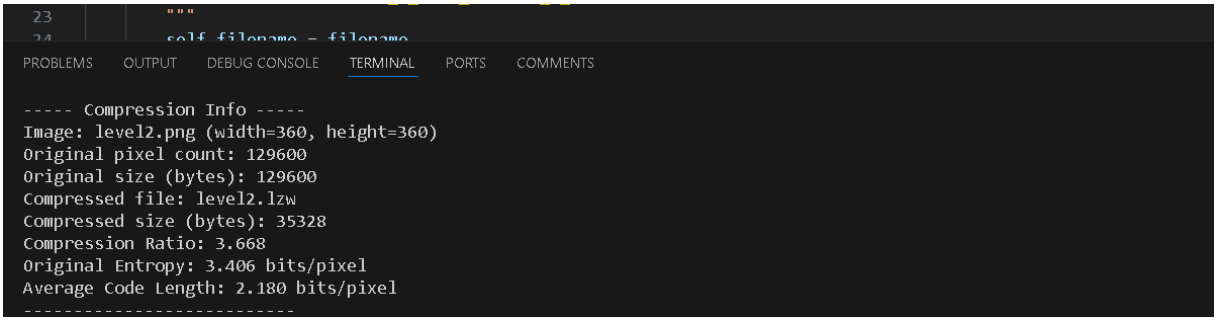
- **Compressed Output (Encoded Integer Codes):**

  [84, 79, 66, 256, 258, 257, 66]*)* level1.lzw

- **Decompressed Output (Restored Text):**

  "Lorem Ipsum is simply dummy…" level1_decompressed.txt

Test 2 – Level 2: Grayscale Image Compression



- **Input File:**

  level2.png

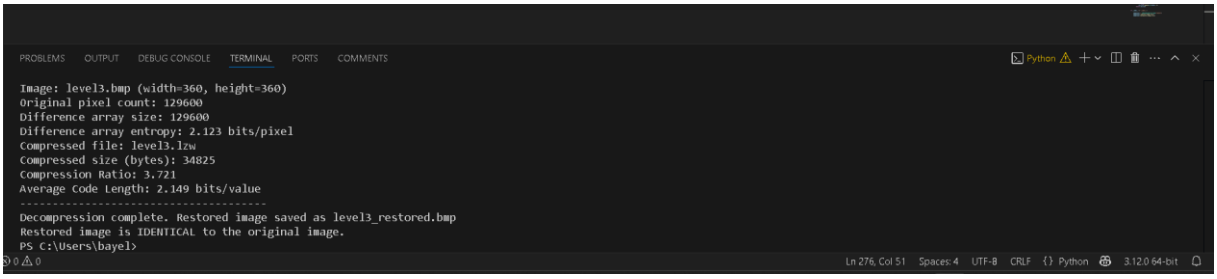
- **Compressed Output:**

  level2.lzw

- **Decompressed Output (Restored Image):**

  level2_restored.png

Test 3 – Level 3: Difference-Based Grayscale Compression



- **Input File:**

- **Compressed Output:**

  level3.lzw

- **Decompressed Output (Restored Image):**

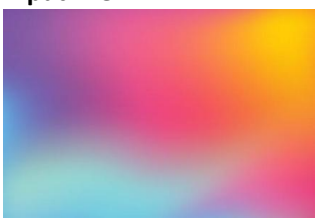
Test 4 – Level 4: RGB Image Compression

```
        self.filename = filename

        self.codelengths = {'R': 0, 'G': 0, 'B': 0}
        self.width = 0
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS                Python

Original size (bytes): 750924
Compressed file: level4.lzw
Compressed size (bytes): 252950
Compression Ratio: 2.969
--- Channel Entropies ---
R Entropy: 6.967 bits, G Entropy: 6.984 bits, B Entropy: 7.578 bits
--- Avg Code Length (bits/pixel) ---
R: 2.638, G: 2.629, B: 2.816
------------------------------------
Decompression complete. Restored image saved as level4_restored.bmp
Restored image is IDENTICAL to the original image.
PS C:\Users\bayel>
⚠ 0                                  Ln 329, Col 51  Spaces: 4  UTF-8  CRLF  {} Python  3.12.0 64-bit
```

- **Input File:**



- **Compressed Output:**

level4.lzw

- **Decompressed Output (Restored Image):**

Test 5 – Level 5: Difference-Based Color Image Compression

```
322             raise ValueError("The encoded data is not padded properly!")
323         b = bytearray()
324         for i in range(0, len(padded_encoded_data), 8):
325             byte = padded_encoded_data[i:i+8]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Original size (bytes): 750924
Compressed file: level5.lzw
Compressed size (bytes): 141964
Compression Ratio: 5.290
--- Channel Difference Entropies ---
R-diff Entropy: 1.445, G-diff Entropy: 1.346, B-diff Entropy: 1.440
--- Avg Code Length (bits/pixel) ---
R: 1.510, G: 1.505, B: 1.521
------------------------------------------------------------
Decompression complete. Restored image saved as level5_restored.bmp
Restored image is IDENTICAL to the original image.
PS C:\Users\bayel>
```

Ln 342, Col 4 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit

- **Input File:**

- **Compressed Output:**

level5.lzw

- **Decompressed Output (Restored Image):**

Test 6 – Level 6: Final GUI Compression (All Methods Integrated)

---

7. List of Achievements – What Did You Learn?

During this project, we gained valuable experience in **algorithm design, image processing, and software development**. More importantly, it was an opportunity to **push our limits and overcome technical challenges**.

. **Overcoming Technical Challenges and Developing Problem-Solving Skills**

After completing the **COMP100: Introduction to Python** course at our university, working on this project was initially a challenge. We spent hours refining our Python skills, debugging issues, and troubleshooting errors.

The biggest contribution of this course and project was that it **forced us to research, learn independently, and solve problems on our own**.

Above all, this was our first **product-oriented** project, and we worked with excitement to deliver a functional system. There were moments when **RGB channels caused errors, and our GUI wouldn't launch**, but today, we have successfully submitted our final product.

- **Understanding the LZW Algorithm and Dictionary-Based Compression**
  We learned how **Lempel-Ziv-Welch (LZW) compression works** and how dictionary-based compression efficiently reduces data size while maintaining lossless quality.
- **Applying Difference-Based Encoding in Image Compression**
  We explored the importance of **difference encoding** for grayscale and RGB images and how **entropy reduction** improves compression efficiency.

- **Developing a GUI in Python and Creating a User-Friendly Interface**
  Working with **Tkinter**, we designed an interactive **Graphical User Interface (GUI)** to allow users to test different compression techniques.

- **Analyzing Performance of Different Compression Techniques**
  By implementing multiple compression strategies, we were able to compare their **effectiveness using metrics like compression ratio, entropy, and average code length**.

- **Project Evolution and Continuous Improvement**
  Throughout the development process, the project evolved as we learned new techniques and saw improvements from our peers. Over time, our understanding of compression algorithms deepened, and our implementation became more efficient. *(Below, we will also include the evolution of our project and development phases.)*