Pip install

```
!pip install pycryptodome
```

```
Requirement already satisfied: pycryptodome in /usr/local/lib/python3.12/dist-packages (3.23.0)
```

Import Required Libraries

```
import base64
from Crypto.Cipher import AES
import numpy as np
import cv2
from PIL import Image
from skimage.metrics import structural_similarity as ssim_fn  # <-- renamed for safe usage
import matplotlib.pyplot as plt
import os
```

Encryption Functions

```
def pad(data):
    pad_len = 16 - len(data) % 16
    return data + bytes([pad_len] * pad_len)

def unpad(data):
    pad_len = data[-1]
    return data[:-pad_len]

def aes_encrypt(message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    padded_msg = pad(message.encode())
    encrypted = cipher.encrypt(padded_msg)
    return base64.b64encode(encrypted).decode()

def aes_decrypt(encrypted_message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    decoded = base64.b64decode(encrypted_message)
    decrypted = cipher.decrypt(decoded)
    return unpad(decrypted).decode()
```

```
def xor_encrypt(message, key):
    return ''.join(chr(ord(c) ^ ord(key[i % len(key)])) for i, c in enumerate(message))

def xor_decrypt(encrypted_message, key):
    return xor_encrypt(encrypted_message, key)
```

LSB Embedder Functions

```
def embed_message(image_path, message, output_path):
    img = Image.open(image_path).convert('L')
    data = np.array(img)
    flat_data = data.flatten()
    binary_msg = ''.join(format(ord(c), '08b') for c in message)
    if len(binary_msg) > len(flat_data):
        raise ValueError("Message too long to embed in image.")
    for i in range(len(binary_msg)):
        flat_data[i] = (flat_data[i] & 0xFE) | int(binary_msg[i])
    new_data = flat_data.reshape(data.shape)
    Image.fromarray(new_data.astype(np.uint8)).save(output_path)

def extract_message(image_path, msg_length):
    img = Image.open(image_path).convert('L')
    data = np.array(img).flatten()
    bits = [str(data[i] & 1) for i in range(msg_length * 8)]
    chars = [chr(int(''.join(bits[i:i+8]), 2)) for i in range(0, len(bits), 8)]
    return ''.join(chars)
```

Quality Metrics Functions

```
def compute_psnr(original_path, modified_path):
    original = cv2.imread(original_path, cv2.IMREAD_GRAYSCALE)
    modified = cv2.imread(modified_path, cv2.IMREAD_GRAYSCALE)
    mse = np.mean((original - modified) ** 2)
    if mse == 0:
        return float('inf')
    PIXEL_MAX = 255.0
    return 20 * np.log10(PIXEL_MAX / np.sqrt(mse))

def compute_ssim(original_path, modified_path):
    original = cv2.imread(original_path, cv2.IMREAD_GRAYSCALE)
```

```
        modified = cv2.imread(modified_path, cv2.IMREAD_GRAYSCALE)
        return ssim_fn(original, modified)
```

Visualizer Function

```
def plot_metrics(lengths, psnr_values, ssim_values):
    plt.figure(figsize=(10, 5))
    plt.plot(lengths, psnr_values, label='PSNR')
    plt.plot(lengths, ssim_values, label='SSIM')
    plt.xlabel('Message Length')
    plt.ylabel('Metric Value')
    plt.title('Capacity vs Quality Trade-Off')
    plt.legend()
    plt.grid(True)
    plt.savefig('quality_tradeoff.png')
    plt.show()
```

Main Workflow

Mounting Google Drive (Optional):

This step is necessary if your cover image is stored in Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
message = "THIS IS THE ENCRYPTED MESSAGE"
encryption_type = "aes"
aes_key = b"abcdefghijklmnop"
xor_key = "secretkey"
cover_image_path = "/content/drive/MyDrive/project/DWS PROJECT/images/random.jpg"
stego_image_path = "/content/drive/MyDrive/project/DWS PROJECT/images/Stego_random.png"
```

1. Encrypt the Message

```
message = "THIS IS THE ENCRYPTED MESSAGE"
encryption_type = "aes"
aes_key = b"abcdefghijklmnop"
xor_key = "secretkey"

if encryption_type == "aes":
    ciphertext = aes_encrypt(message, aes_key)
    decrypt_func = lambda ct: aes_decrypt(ct, aes_key)
elif encryption_type == "xor":
    ciphertext = xor_encrypt(message, xor_key)
    decrypt_func = lambda ct: xor_decrypt(ct, xor_key)
else:
    raise ValueError("encryption_type must be 'aes' or 'xor'.")

print("Ciphertext to embed:", ciphertext)
```

```
Ciphertext to embed: ZygdxpnG/DfOi4lfrwvvlAn24UmTnhUvJKaNo/XxOxU=
```

2. Embed Ciphertext

```
embed_message(cover_image_path, ciphertext, stego_image_path)
print("Stego image saved as:", stego_image_path)
```

```
Stego image saved as: /content/drive/MyDrive/project/DWS PROJECT/images/Stego_random.png
```

3. Compute Image Quality Metrics

```
psnr = compute_psnr(cover_image_path, stego_image_path)
ssim_val = compute_ssim(cover_image_path, stego_image_path)
print(f"PSNR: {psnr:.2f}")
print(f"SSIM: {ssim_val:.4f}")
```

```
PSNR: 74.76
SSIM: 1.0000
```

4. Extract and Decrypt Message

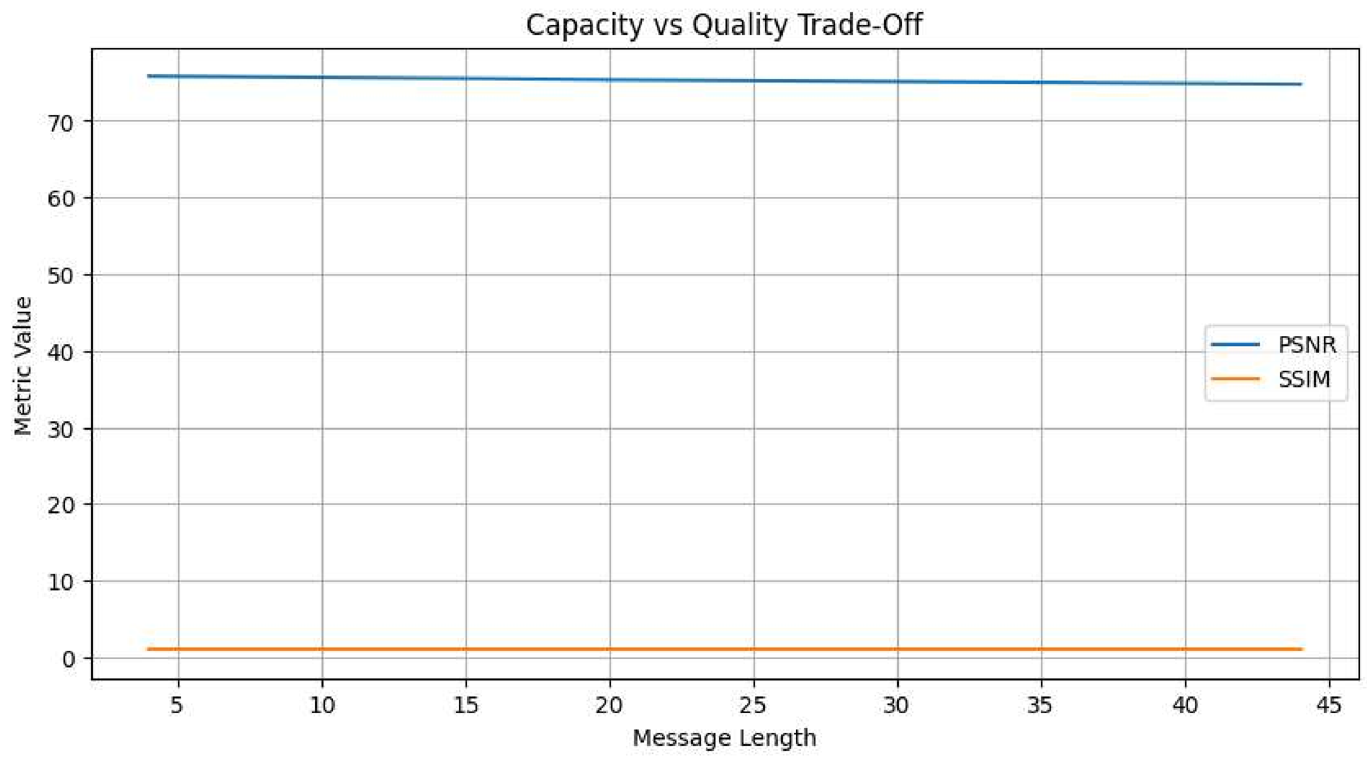
```
msglen = len(ciphertext)
extracted = extract_message(stego_image_path, msglen)
recovered = decrypt_func(extracted)
print("Extracted and decrypted message:", recovered)
```

```
Extracted and decrypted message: THIS IS THE ENCRYPTED MESSAGE
```

5. Optional: Capacity vs Quality Plot

```
# 5.
lengths, psnr_values, ssim_values = [], [], []
step = max(1, msglen // 10)
for l in range(step, msglen + 1, step):
    temp_path = "/content/drive/MyDrive/project/DWS PROJECT/images/temp_stego.png"
    embed_message(cover_image_path, ciphertext[:l], temp_path)
    lengths.append(l)
    psnr_values.append(compute_psnr(cover_image_path, temp_path))
    ssim_values.append(compute_ssim(cover_image_path, temp_path))
    os.remove(temp_path)

plot_metrics(lengths, psnr_values, ssim_values)
print("Trade-off plot saved as 'quality_tradeoff.png'.")
```



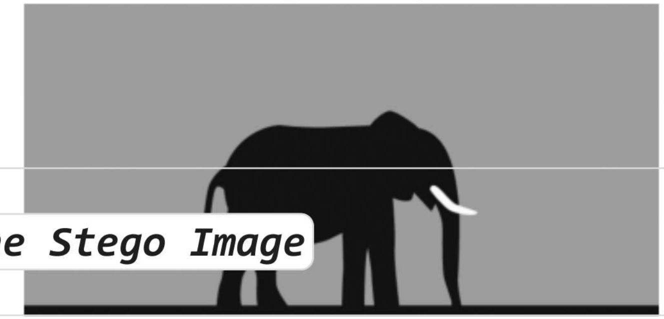
```
Trade-off plot saved as 'quality_tradeoff.png'.
```

The Original Image

```
original_image = cv2.imread(cover_image_path, cv2.IMREAD_GRAYSCALE)

plt.figure(figsize=(5, 5))
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.show()
```

Original Image



The Stego Image

```
# Load and display
stego_image = cv2.imread(stego_image_path, cv2.IMREAD_GRAYSCALE)

plt.figure(figsize=(5, 5))
plt.imshow(stego_image, cmap='gray')
plt.title('Stego Image')
plt.axis('off')
plt.show()
```

Stego Image