Table 4 and Fig. 26 Data Analysis

The data stored in

/content/drive/My Drive/ShanShui/

1. guo_xi.jpg

2. Picture27a.jpg

3. Picture27b.jpg

4...

Please see the Zipped file named: ShanShui-20250813T111616Z-1-001.zip

```
from google.colab import drive
from PIL import Image
import matplotlib.pyplot as plt
import cv2
import numpy as np
import os
from skimage.metrics import structural_similarity as ssim
import pandas as pd
import math

# เชื่อมต่อ Google Drive
drive.mount('/content/drive')


# Install required libraries
!pip install Pillow matplotlib opencv-python numpy scikit-image

# Define the directory containing the images
drive_dir = '/content/drive/My Drive/ShanShui'

# Define the list of Shanshui backbone candidates
backbone_candidates = ['Picture5.jpg', 'Picture22.jpg', 'Picture24.jpg']

# Define a reference image (example: guo_xi.jpg)
reference_image_name = 'guo_xi.jpg'
reference_image_path = os.path.join(drive_dir, reference_image_name)

# Define a target size for resizing (adjust as needed)
target_size = (800, 600)  # width, height

# Function to resize and load images
def load_and_resize_image(image_path, target_size):
    try:
        img = Image.open(image_path).convert("RGB") # Ensure image is in RGB format
        img_resized = img.resize(target_size)
        return img_resized
    except FileNotFoundError:
        print(f"Error: Image not found at {image_path}")
        return None
```

```
    except Exception as e:
        print(f"Error loading or resizing image {image_path}: {e}")
        return None

# Load and resize the reference image
reference_img_resized_pil = load_and_resize_image(reference_image_path, target_size)

if reference_img_resized_pil:
    print(f"Loaded and resized reference image: {reference_image_name}")

    # Load, resize, and store backbone candidates
    resized_candidates_pil = {}
    print("\nLoading and resizing backbone candidates:")
    for img_name in backbone_candidates:
        img_path = os.path.join(drive_dir, img_name)
        img_resized = load_and_resize_image(img_path, target_size)
        if img_resized:
            resized_candidates_pil[img_name] = img_resized
            print(f"  - {img_name}")

    # Display the resized images for comparison
    if resized_candidates_pil:
        fig, axes = plt.subplots(1, len(resized_candidates_pil) + 1, figsize=(20, 5))
        axes[0].imshow(reference_img_resized_pil)
        display_ref_name = reference_image_name
        if reference_image_name == "guo_xi.jpg":
          display_ref_name = "Guo Xi's Shanshui"
        axes[0].set_title(f'Reference: {display_ref_name}\n({target_size[0]}x{target_size[1]})')
        axes[0].axis('off')

        for i, (name, img) in enumerate(resized_candidates_pil.items()):
            axes[i+1].imshow(img)
            display_name = name
            if name == "Picture5.jpg":
              display_name = "Ours, LoRA Fine-tuned Backbone (Fig. 5)"
            elif name == "Picture22.jpg":
              display_name = "LoRA 2nd Session Backbone (Fig. 22)"
            elif name == "Picture24.jpg":
              display_name = "LoRA 3rd Session Backbone (Fig. 24)"
            axes[i+1].set_title(f'Candidate: {display_name}\n({target_size[0]}x{target_size[1]})')
            axes[i+1].axis('off')

        plt.tight_layout()
        plt.show()

        # Convert PIL images to OpenCV format (numpy arrays) for metric calculations
        reference_img_resized_cv2 = cv2.cvtColor(np.array(reference_img_resized_pil), cv2.COLOR_RGB2BGR)
        resized_candidates_cv2 = {name: cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR) for name, img in resized_candidates_pil.items()}

        # --- Image Metric Calculations ---
```

```python
# Function to calculate MSE
def calculate_mse(img1, img2):
    # The images must have the same dimension
    if img1.shape != img2.shape:
        raise ValueError("Images must have the same dimensions for MSE calculation.")
    err = np.sum((img1.astype("float") - img2.astype("float")) ** 2)
    err /= float(img1.shape[0] * img1.shape[1] * img1.shape[2])
    return err

# Function to calculate PSNR
def calculate_psnr(img1, img2):
    mse = calculate_mse(img1, img2)
    if mse == 0:  # MSE is zero means no noise, the images are identical
        return float('inf')
    max_pixel = 255.0  # For 8-bit images
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr

# Function to calculate SSIM
def calculate_ssim(img1, img2):
    # SSIM requires images to be grayscale or have the same channel count
    # Ensure channel count matches or convert to grayscale
    if img1.shape[-1] == 3 and img2.shape[-1] == 3:
        # Calculate SSIM on each channel and take the average
        ssim_value = ssim(img1, img2, channel_axis=-1) # Use channel_axis for multichannel images
    elif img1.shape == img2.shape:
        # Assume grayscale
        ssim_value = ssim(img1, img2)
    else:
        raise ValueError("Images must have compatible dimensions and channels for SSIM.")
    return ssim_value


# Function to calculate Color Histogram Cosine Distance
def calculate_color_histogram_cosine_distance(img1, img2):
    # Calculate color histograms
    hist1 = cv2.calcHist([img1], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    hist2 = cv2.calcHist([img2], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    # Normalize histograms
    hist1 = cv2.normalize(hist1, hist1).flatten()
    hist2 = cv2.normalize(hist2, hist2).flatten()
    # Calculate cosine similarity
    cosine_similarity = np.dot(hist1, hist2) / (np.linalg.norm(hist1) * np.linalg.norm(hist2))
    # Cosine distance is 1 - cosine similarity
    cosine_distance = 1 - cosine_similarity
    return cosine_distance

# Function to calculate Edge Feature Cosine Distance (using Canny edges)
def calculate_edge_feature_cosine_distance(img1, img2):
```

```
    # Convert to grayscale
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    # Apply Canny edge detection
    edges1 = cv2.Canny(gray1, 100, 200)
    edges2 = cv2.Canny(gray2, 100, 200)
    # Flatten edge images to vectors
    edges1_flat = edges1.flatten().astype(float)
    edges2_flat = edges2.flatten().astype(float)
    # Calculate cosine similarity
    # Handle cases where one or both edge images are all zeros (no edges detected)
    norm1 = np.linalg.norm(edges1_flat)
    norm2 = np.linalg.norm(edges2_flat)

    if norm1 == 0 and norm2 == 0:
        cosine_similarity = 1.0 # Identical (no edges)
    elif norm1 == 0 or norm2 == 0:
        cosine_similarity = 0.0 # One has edges, the other doesn't
    else:
        cosine_similarity = np.dot(edges1_flat, edges2_flat) / (norm1 * norm2)

    # Cosine distance is 1 - cosine similarity
    cosine_distance = 1 - cosine_similarity
    return cosine_distance


# Store results
results = {}

print("\n--- Calculating Image Similarity Metrics ---")
for name, img_candidate_cv2 in resized_candidates_cv2.items():
    print(f"\nComparing {name} with {reference_image_name}:")
    # Ensure the candidate image is also in BGR format (OpenCV default)
    # (already handled during conversion)

    # Calculate metrics
    mse = calculate_mse(reference_img_resized_cv2, img_candidate_cv2)
    psnr = calculate_psnr(reference_img_resized_cv2, img_candidate_cv2)
    ssim_val = calculate_ssim(reference_img_resized_cv2, img_candidate_cv2)
    color_hist_dist = calculate_color_histogram_cosine_distance(reference_img_resized_cv2, img_candidate_cv2)
    edge_feat_dist = calculate_edge_feature_cosine_distance(reference_img_resized_cv2, img_candidate_cv2)

    print(f"  MSE: {mse:.4f}")
    print(f"  PSNR: {psnr:.4f} dB")
    print(f"  SSIM: {ssim_val:.4f}")
    print(f"  Color Histogram Cosine Distance: {color_hist_dist:.4f}")
    print(f"  Edge Feature Cosine Distance: {edge_feat_dist:.4f}")

    # Store results
    results[name] = {
```

```
        'MSE': mse,
        'PSNR': psnr,
        'SSIM': ssim_val,
        'Color_Hist_Cosine_Dist': color_hist_dist,
        'Edge_Feat_Cosine_Dist': edge_feat_dist
    }

# --- Normalize and Prepare for Spider Graph ---
# We want all values between 0 and 1 for the spider graph.
# For SSIM and PSNR (similarity metrics), higher is better, scale directly.
# For MSE, Color Hist Distance, Edge Feat Distance (distance metrics), lower is better.
# We need to invert them or scale them such that higher value means more similar.
# Option: 1 - distance for distance metrics.
# Option: Normalize SSIM (already 0-1), PSNR needs scaling (log scale?), MSE needs scaling.
# Let's normalize all metrics to be between 0 and 1, where 1 means perfect similarity to the reference.

normalized_results = {}

# Collect all values for normalization
all_mse = [res['MSE'] for res in results.values()]
all_psnr = [res['PSNR'] for res in results.values() if res['PSNR'] != float('inf')] # Exclude inf for normalization
all_ssim = [res['SSIM'] for res in results.values()] # SSIM is already 0-1
all_color_dist = [res['Color_Hist_Cosine_Dist'] for res in results.values()]
all_edge_dist = [res['Edge_Feat_Cosine_Dist'] for res in results.values()]

# Simple min-max scaling for metrics that are not already 0-1
def min_max_scale(values, min_val, max_val):
    if max_val - min_val == 0:
        return [0.5] * len(values) # Avoid division by zero, assign a neutral value
    return [(v - min_val) / (max_val - min_val) for v in values]

# Normalize MSE (lower is better, so invert after scaling)
min_mse, max_mse = min(all_mse), max(all_mse)
scaled_mse = min_max_scale(all_mse, min_mse, max_mse)
normalized_mse = [1 - s for s in scaled_mse] # Inverted: 1 is perfect similarity (0 MSE)

# Normalize PSNR (higher is better)
min_psnr = min(all_psnr) if all_psnr else 0
max_psnr = max(all_psnr) if all_psnr else 1 # Avoid division by zero if only one image or all inf
scaled_psnr = min_max_scale(all_psnr, min_psnr, max_psnr)
# Handle inf PSNR: assign 1 (perfect similarity)
normalized_psnr = [1.0 if results[list(results.keys())[i]]['PSNR'] == float('inf') else scaled_psnr[i] for i in range(len(all_psnr))]


# Normalize SSIM (already 0-1, higher is better) - no scaling needed

# Normalize Color Hist Distance (lower is better, so invert)
min_color_dist, max_color_dist = min(all_color_dist), max(all_color_dist)
scaled_color_dist = min_max_scale(all_color_dist, min_color_dist, max_color_dist)
normalized_color_dist = [1 - s for s in scaled_color_dist] # Inverted: 1 is perfect similarity (0 distance)
```

```
# Normalize Edge Feat Distance (lower is better, so invert)
min_edge_dist, max_edge_dist = min(all_edge_dist), max(all_edge_dist)
scaled_edge_dist = min_max_scale(all_edge_dist, min_edge_dist, max_edge_dist)
normalized_edge_dist = [1 - s for s in scaled_edge_dist] # Inverted: 1 is perfect similarity (0 distance)

# Populate normalized results dictionary
metric_names = ['Normalized_MSE', 'SSIM', 'Normalized_PSNR', 'Color_Hist_Similarity', 'Edge_Feat_Similarity']
# Rename distance metrics to similarity for clarity in the spider graph
normalized_results_list = []
for i, name in enumerate(results.keys()):
    normalized_results[name] = {
        'Normalized_MSE': normalized_mse[i],
        'SSIM': all_ssim[i], # SSIM is already similarity
        'Normalized_PSNR': normalized_psnr[i],
        'Color_Hist_Similarity': normalized_color_dist[i], # 1 - distance = similarity
        'Edge_Feat_Similarity': normalized_edge_dist[i]   # 1 - distance = similarity
    }
    normalized_results_list.append([
        normalized_mse[i],
        all_ssim[i],
        normalized_psnr[i],
        normalized_color_dist[i],
        normalized_edge_dist[i]
    ])

# Convert to DataFrame for easier handling
df_normalized = pd.DataFrame.from_dict(normalized_results, orient='index')

print("\n--- Normalized Similarity Metrics (0-1, 1 is perfect similarity) ---")
print(df_normalized)

# --- Plotting Spider Graph ---
categories = list(df_normalized.columns)
N = len(categories)

# Calculate angle for each axis
angles = [n / float(N) * 2 * np.pi for n in range(N)]
angles += angles[:1] # Complete the circle

fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=dict(polar=True))

# Plot data
for i, (candidate_name, row) in enumerate(df_normalized.iterrows()):
    values = row.values.flatten().tolist()
    values += values[:1] # Complete the circle
    # Use display names for the legend
    display_name = candidate_name
    if candidate_name == "Picture5.jpg":
      display_name = "Ours, LoRA Fine-tuned Backbone (Fig. 5)"
```

```
            elif candidate_name == "Picture22.jpg":
                display_name = "LoRA 2nd Session Backbone (Fig. 22)"
            elif candidate_name == "Picture24.jpg":
                display_name = "LoRA 3rd Session Backbone (Fig. 24)"
            ax.plot(angles, values, linewidth=2, linestyle='solid', label=display_name)
            ax.fill(angles, values, alpha=0.25)

        # Set labels and title
        ax.set_thetagrids(np.degrees(angles[:-1]), categories)
        ax.set_title("Image Similarity Metrics vs. Reference Image", va='bottom')
        ax.grid(True)
        ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))

        # Set y-limits for the scale 0 to 1
        ax.set_ylim(0, 1)
        ax.set_yticks(np.arange(0, 1.1, 0.2)) # Set y-ticks at 0, 0.2, 0.4, ... 1.0
        ax.set_yticklabels([f'{y:.1f}' for y in np.arange(0, 1.1, 0.2)]) # Label y-ticks

        plt.show()

        # --- Summarize Meaning of Spider Graph in a Table ---
        print("\n--- Summary of Spider Graph Interpretation ---")
        summary_data = {
            'Metric': metric_names,
            'Interpretation (Higher Value = More Similar)': [
                'Normalized Mean Squared Error (Lower MSE -> Higher Similarity)',
                'Structural Similarity Index (Higher SSIM -> More Similar)',
                'Normalized Peak Signal-to-Noise Ratio (Higher PSNR -> More Similar)',
                'Color Histogram Similarity (Lower distance -> Higher Similarity)',
                'Edge Feature Similarity (Lower distance -> Higher Similarity)'
            ],
            'Range': ['0 - 1', '0 - 1', '0 - 1', '0 - 1', '0 - 1']
        }
        df_summary = pd.DataFrame(summary_data)
        print(df_summary.to_markdown(index=False))

        # --- Explain Interpretation in English Text ---
        print("\n--- Interpretation of the Spider Graph (English Text) ---")
        print("The spider graph visualizes the similarity of the candidate images (Picture5.jpg, Picture24.jpg, Picture26.jpg) to the reference image (guo_xi.jpg) based on several image metrics.")
        print("Each axis represents a different similarity metric, normalized to a range between 0 and 1.")
        print("- A value closer to 1 on any axis indicates higher similarity to the reference image according to that specific metric.")
        print("- A value closer to 0 indicates lower similarity (or higher difference).")
        print("\nHere's what each metric represents:")
        print("- **Normalized MSE (Mean Squared Error):** Measures the average squared difference between the pixel values of the two images. The value is inverted and normalized, so a higher value means lower error and thus higher similarity.")
        print("- **SSIM (Structural Similarity Index):** Measures similarity in terms of structural information, luminance, and contrast. A higher SSIM value means the images are perceived as more similar by the human visual system.")
```

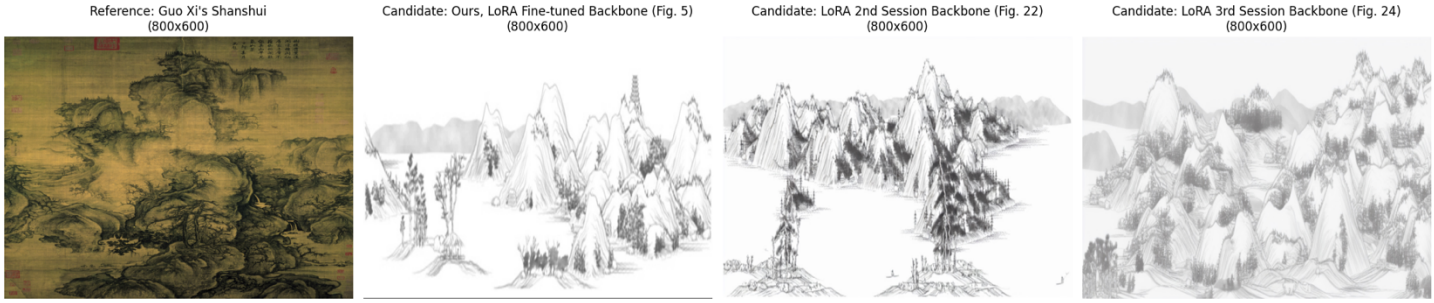
```
        print("- **Normalized PSNR (Peak Signal-to-Noise Ratio):** Measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. The value is normalized, so a higher value generally indicates higher quality and similarity, often used for assessing lossy compression.")
        print("- **Color Histogram Similarity:** Measures the similarity of the distribution of colors in the two images using the cosine similarity of their color histograms. A higher value means the images have a more similar overall color composition.")
        print("- **Edge Feature Similarity:** Measures the similarity of the prominent edge patterns in the two images, calculated using the cosine similarity of flattened Canny edge maps. A higher value suggests similar structural outlines and features.")
        print("\nBy observing the shape and area covered by each candidate's line on the spider graph, you can compare their overall similarity profile to the reference image across multiple dimensions. A candidate with a larger area covered by its line and values closer to the outer edge (1) is generally more similar to the reference image across the evaluated metrics.")
        print("This helps identify which candidate images share more visual characteristics with the reference painting, providing quantitative insights into their resemblance.")


    else:
        print("No backbone candidates were successfully loaded.")

else:
    print(f"Could not load the reference image: {reference_image_name}. Please check the path.")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (11.3.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.12.0.88)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dist-packages (0.25.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1.16.1)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (3.5)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2025.6.11)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (0.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Loaded and resized reference image: guo_xi.jpg

Loading and resizing backbone candidates:
  - Picture5.jpg
  - Picture22.jpg
  - Picture24.jpg
```



--- Calculating Image Similarity Metrics ---

Comparing Picture5.jpg with guo_xi.jpg:
MSE: 26329.5265
PSNR: 3.9264 dB
SSIM: 0.2140
Color Histogram Cosine Distance: 0.9998
Edge Feature Cosine Distance: 0.9001

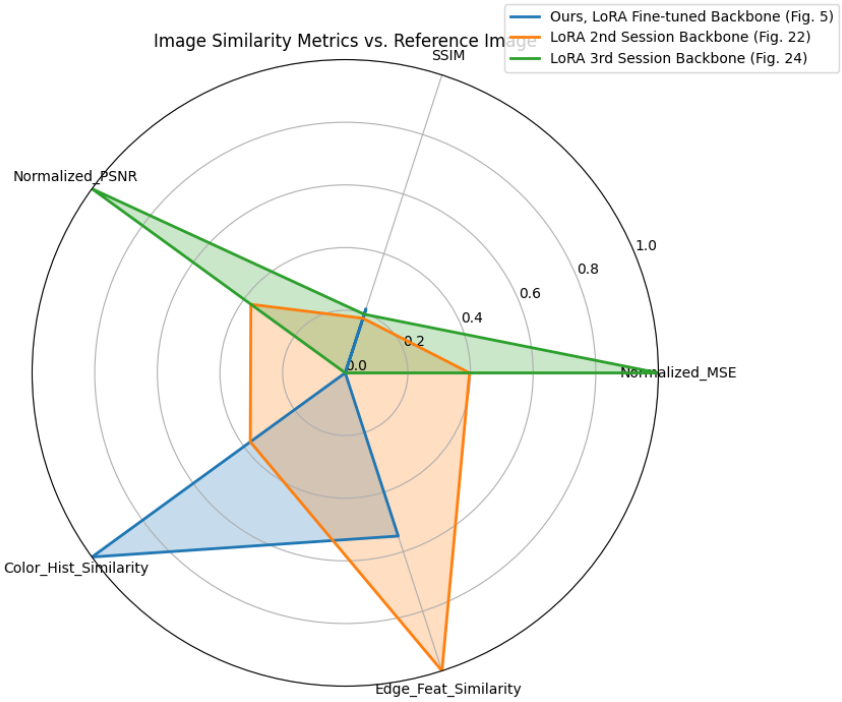
Comparing Picture22.jpg with guo_xi.jpg:
MSE: 24373.5166
PSNR: 4.2616 dB
SSIM: 0.1833
Color Histogram Cosine Distance: 0.9999
Edge Feature Cosine Distance: 0.8906

Comparing Picture24.jpg with guo_xi.jpg:
MSE: 21400.7392
PSNR: 4.8265 dB
SSIM: 0.1967
Color Histogram Cosine Distance: 1.0000
Edge Feature Cosine Distance: 0.9116

--- Normalized Similarity Metrics (0-1, 1 is perfect similarity) ---

	Normalized_MSE	SSIM	Normalized_PSNR	\
Picture5.jpg	0.000000	0.213994	0.000000	
Picture22.jpg	0.396854	0.183331	0.372439	
Picture24.jpg	1.000000	0.196690	1.000000	

	Color_Hist_Similarity	Edge_Feat_Similarity
Picture5.jpg	1.000000	0.547626
Picture22.jpg	0.374746	1.000000
Picture24.jpg	0.000000	0.000000



--- Summary of Spider Graph Interpretation ---

Metric	Interpretation (Higher Value = More Similar)	Range
Normalized_MSE	Normalized Mean Squared Error (Lower MSE -> Higher Similarity)	0 - 1
SSIM	Structural Similarity Index (Higher SSIM -> More Similar)	0 - 1
Normalized_PSNR	Normalized Peak Signal-to-Noise Ratio (Higher PSNR -> More Similar)	0 - 1
Color_Hist_Similarity	Color Histogram Similarity (Lower distance -> Higher Similarity)	0 - 1
Edge_Feat_Similarity	Edge Feature Similarity (Lower distance -> Higher Similarity)	0 - 1

--- Interpretation of the Spider Graph (English Text) ---

The spider graph visualizes the similarity of the candidate images (Picture5.jpg, Picture24.jpg, Picture26.jpg) to the reference image (guo_xi.jpg) based on several image metrics.

Each axis represents a different similarity metric, normalized to a range between 0 and 1.

- A value closer to 1 on any axis indicates higher similarity to the reference image according to that specific metric.
- A value closer to 0 indicates lower similarity (or higher difference).

Here's what each metric represents:

- **Normalized MSE (Mean Squared Error):** Measures the average squared difference between the pixel values of the two images. The value is inverted and normalized, so a higher value means lower error and thus higher similarity.
- **SSIM (Structural Similarity Index):** Measures similarity in terms of structural information, luminance, and contrast. A higher SSIM value means the images are perceived as more similar by the human visual system.
- **Normalized PSNR (Peak Signal-to-Noise Ratio):** Measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. The value is normalized, so a higher value generally indicates higher quality and similarity, often used for assessing lossy compression.
- **Color Histogram Similarity:** Measures the similarity of the distribution of colors in the two images using the cosine similarity of their color histograms. A higher value means the images have a more similar overall color composition.
- **Edge Feature Similarity:** Measures the similarity of the prominent edge patterns in the two images, calculated using the cosine similarity of flattened Canny edge maps. A higher value suggests similar structural outlines and features.

By observing the shape and area covered by each candidate's line on the spider graph, you can compare their overall similarity profile to the reference image across multiple dimensions. A candidate with a larger area covered by its line and values closer to the outer edge (1) is generally more similar to the reference image across the evaluated metrics.

This helps identify which candidate images share more visual characteristics with the reference painting, providing quantitative insights into their resemblance.