import os

import json

import numpy as np

import pandas as pd

# Import methods from different analysis modules

from EntityAnalysis import process\_json\_files, get\_Entity\_Accuracy, get\_Entity\_Precision, get\_Entity\_Recall, get\_Entity\_F1

from ScoreAnalysis import get\_sc, get\_sd, extract\_scores

from StyleAnalysis import get\_ass

from TextAnalysis import get\_tar, get\_ts, process\_directory as process\_text\_analysis

# Function to load JSON files

def load\_json\_data(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return json.load(f)

except FileNotFoundError:

print(f"File not found: {file\_path}")

return None

# Process entity analysis and save results to Excel

def process\_entity\_analysis(directory\_path, output\_file):

entity\_results = []

for i in range(1, 21):

file\_path = os.path.join(directory\_path, f'{i}.jpg\_labels.json')

print(file\_path)

TP, FP, FN, MR = process\_json\_files(file\_path)

print("TP", TP, "FP", FP, "FN", FN, "MR", MR)

Accuracy = get\_Entity\_Accuracy(TP, FP, FN, MR)

Precision = get\_Entity\_Precision(TP, FP, MR)

Recall = get\_Entity\_Recall(TP, FN, MR)

F1 = get\_Entity\_F1(Precision, Recall)

print("Accuracy", Accuracy, "Precision", Precision, "Recall", Recall, "F1", F1)

entity\_results.append({

'file': f'{i}.jpg\_labels.json',

'Accuracy': Accuracy,

'Precision': Precision,

'Recall': Recall,

'F1': F1

})

entity\_results\_df = pd.DataFrame(entity\_results)

entity\_results\_df.to\_excel(output\_file, index=False)

print(f"Entity Analysis Results have been saved to {output\_file}")

# Process score analysis and save results to Excel

def process\_score\_analysis(score\_review\_dir, output\_file):

output\_file\_scores = "SC\_SD\_Sequences.xlsx"

extract\_scores(score\_review\_dir, output\_file\_scores)

dimensions = [

"Realistic", "Deformation", "Imagination", "Color Richness",

"Color Contrast", "Line Combination", "Line Texture",

"Picture Organization", "Transformation"

]

score\_results = []

scores\_df = pd.read\_excel(output\_file\_scores)

for dimension in dimensions:

dim\_scores = scores\_df[scores\_df['dimension'] == dimension]

original\_scores = dim\_scores['original']

current\_scores = dim\_scores['current']

# 计算SC和SD

sc\_value = get\_sc(original\_scores, current\_scores)

sd\_value = get\_sd(original\_scores, current\_scores)

score\_results.append({

'dimension': dimension,

'SC': sc\_value,

'SD': sd\_value

})

score\_results\_df = pd.DataFrame(score\_results)

score\_results\_df.to\_excel(output\_file, index=False)

print(f"Score Analysis Results have been saved to {output\_file}")

# Process art style sensitivity (ASS) and save results to Excel

def process\_style\_analysis(folder\_path, output\_file):

get\_ass(folder\_path, output\_file)

# Process text analysis (TAR & TS) and save results to Excel

def process\_text\_analysis\_main(score\_comment\_dir, suggestion\_dir, output\_file\_tar\_rev, output\_file\_tar\_sug, output\_file\_ts\_rev,output\_file\_ts\_sug):

process\_text\_analysis(score\_comment\_dir, suggestion\_dir, output\_file\_tar\_rev, output\_file\_tar\_sug, output\_file\_ts\_rev,output\_file\_ts\_sug)

# Main function: call each analysis and save results

if \_\_name\_\_ == "\_\_main\_\_":

# Entity analysis

entity\_directory = 'userActions/Entities'

entity\_output\_file = "Entity\_Results.xlsx"

process\_entity\_analysis(entity\_directory, entity\_output\_file)

# Score analysis

score\_review\_directory = 'userActionsEveryRounds/score\_Review'

score\_output\_file = "SC\_SD\_Results.xlsx"

process\_score\_analysis(score\_review\_directory, score\_output\_file)

# Art style analysis

style\_directory = 'userActions/Entities'

style\_output\_file = "ASS\_Results.xlsx"

process\_style\_analysis(style\_directory, style\_output\_file)

# Text analysis (TAR & TS)

score\_comment\_directory = 'userActionsEveryRounds/score\_Review'

suggestion\_directory = 'userActionsEveryRounds/suggestion'

output\_file\_tar\_rev = "TAR\_Results\_rev.xlsx"

output\_file\_tar\_sug = "TAR\_Results\_sug.xlsx"

output\_file\_ts\_rev = "TS\_Results\_rev.xlsx"

output\_file\_ts\_sug = "TS\_Results\_sug.xlsx"

process\_text\_analysis\_main(score\_comment\_directory, suggestion\_directory,

output\_file\_tar\_rev, output\_file\_tar\_sug,

output\_file\_ts\_rev,output\_file\_ts\_sug)

import base64

import matplotlib.pyplot as plt

import numpy as np

from flask import Flask, request, render\_template, redirect, url\_for, flash, jsonify

from matplotlib import rcParams

from werkzeug.utils import secure\_filename

import os

import openai

from io import BytesIO

import re

import json

import difflib

# # 设置全局字体

rcParams['font.family'] = 'Times New Roman'

rcParams['font.size'] = 12 # 设置字体大小

app = Flask(\_\_name\_\_)

app.secret\_key = os.urandom(24)

print(f'Secret Key: {app.secret\_key}')

app.config['UPLOAD\_FOLDER'] = 'uploads/'

app.config['ALLOWED\_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif'}

# 设置 OpenAI 的 API key 和代理

proxy = "http:

os.environ["http\_proxy"] = proxy

os.environ["https\_proxy"] = proxy

# 读取文本文件中的 API key

with open('key.txt', 'r') as file:

api\_key = file.read().strip()

client = openai.OpenAI(api\_key=api\_key)

# 全局变量存储评分数据

global\_scores = {

'Realistic': 0,

'Deformation': 0,

'Imagination': 0,

'Color Richness': 0,

'Color Contrast': 0,

'Line Combination': 0,

'Line Texture': 0,

'Picture Organization': 0,

'Transformation': 0

}

# 每个文件对应的 round 计数器

round\_counters = {}

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED\_EXTENSIONS']

def encode\_image(image\_path):

with open(image\_path, "rb") as image\_file:

return base64.b64encode(image\_file.read()).decode('utf-8')

def Entity\_Recognition\_Agent(image\_data):

response = client.chat.completions.create(

model="gpt-4o",

temperature=0,

top\_p=1,

messages=[

{

"role": "system",

"content": "Identify and list the objects or features present in the image using descriptive labels. Use simple, clear terms like 'Face', 'Black hair', 'Thick lips', 'Big ears', etc. Ensure that each label is descriptive and that labels are separated by the symbol (';'). For example: Face; Black hair; Thick lips; Big ears;. Also, identify the art style of the image with a label starting with '## Style:'."

},

{

"role": "user",

"content": [

{"type": "image\_url",

"image\_url": {

"url": f"data:image/jpeg;base64,{image\_data}",

"detail": "high"

},

},

],

}

],

max\_tokens=100,

)

response\_content = response.choices[0].message.content

# 分离出风格标签和其他标签

objects = response\_content.split(";")

style\_label = None

cleaned\_objects = []

for obj in objects:

obj = obj.strip()

if obj.startswith("## Style:"):

style\_label = obj[len("## Style:"):].strip() # 去掉标志符和前导空格

elif obj:

cleaned\_objects.append(obj)

print("cleaned\_objects",cleaned\_objects)

print("Style Label:", style\_label)

return cleaned\_objects, style\_label

def create\_Suggestion\_prompt(labels\_data, score\_Review\_data, suggestion\_data, dimension):

print("####create\_Suggestion\_prompt####")

print("labels\_data",labels\_data)

print("score\_Review\_data",score\_Review\_data)

print("suggestion\_data",suggestion\_data)

print("dimension",dimension)

if labels\_data:

# 提取 'original' 并移除 'removed' 中的元素

updated\_Entities = labels\_data["original"]

updated\_Entities = [tag for tag in updated\_Entities if tag not in labels\_data["removed"]]

# 确定插入点为列表的倒数第二个位置，如果列表项少于2个，则插入到列表末尾

insert\_index = max(len(updated\_Entities) - 1, 0) # 确保至少有一个元素，否则插入到开头

# 将所有 'added' 元素插入到计算出的位置

for added\_item in reversed(labels\_data["added"]): # 使用 reversed 保证添加的顺序正确

updated\_Entities.insert(insert\_index, added\_item)

Entities = "; ".join(updated\_Entities)

Entities\_prompt = f"The image contains the following entities: {Entities}. "

else:

Entities\_prompt = ""

if score\_Review\_data == {

"scores": {"original": 0, "current": 0, "initGPTscore": None},

"Reviews": {"original": "", "current": "", "added": "", "removed": ""},

}:

score\_Review\_prompt = ""

else:

scores = f"The current score which the human user submitted for this dimension is {score\_Review\_data['scores']['current']}."

Reviews = f"The current Review which the human user submitted for this dimension is: {score\_Review\_data['Reviews']['current']}"

score\_Review\_prompt = f"{scores} {Reviews} "

if suggestion\_data == {"suggestions": {"original": "", "current": "", "added": "", "removed": ""}}:

suggestions\_prompt = ""

else:

suggestions = f"The suggestion which the human user submitted for this dimension is {suggestion\_data['suggestions']['current']}."

suggestions\_prompt = f"{suggestions}"

print("dimension",dimension)

full\_prompt = f"The suggestion dimension is {dimension}. You may receive some Reviews, score or suggestions feedback from the human user. please consider it more.{Entities\_prompt}{score\_Review\_prompt}{suggestions\_prompt}Need improve dimension is {dimension}\n According to dimension, output a dictionary and don't use special symbols such as line breaks for ease of processing:"+"{\"suggestion\":suggestion for artworks to improve"+" }"

# 调试语句：打印生成的 prompt 和其长度

print(f"Generated prompt for {dimension}:")

print(full\_prompt)

print(f"Prompt length (in characters): {len(full\_prompt)}")

return full\_prompt

def create\_Review\_prompt(labels\_data, score\_Review\_data, dimension):

print("####create\_prompt####")

print("labels\_data",labels\_data)

print("score\_Review\_data",score\_Review\_data)

print("dimension",dimension)

if labels\_data:

# 提取 'original' 并移除 'removed' 中的元素

updated\_Entities = labels\_data["original"]

updated\_Entities = [tag for tag in updated\_Entities if tag not in labels\_data["removed"]]

# 确定插入点为列表的倒数第二个位置，如果列表项少于2个，则插入到列表末尾

insert\_index = max(len(updated\_Entities) - 1, 0) # 确保至少有一个元素，否则插入到开头

# 将所有 'added' 元素插入到计算出的位置

for added\_item in reversed(labels\_data["added"]): # 使用 reversed 保证添加的顺序正确

updated\_Entities.insert(insert\_index, added\_item)

Entities = "; ".join(updated\_Entities)

Entities\_prompt = f"The artworks from child contains the following entities: {Entities}. "

else:

Entities\_prompt = ""

if score\_Review\_data == {

"scores": {"original": 0, "current": 0, "initGPTscore": None},

"Reviews": {"original": "", "current": "", "added": "", "removed": ""},

}:

score\_Review\_prompt = ""

else:

scores = f"The current score which the human user submitted for this dimension is {score\_Review\_data['scores']['current']}."

Reviews = f"The current Review which the human user submitted for this dimension is: {score\_Review\_data['Reviews']['current']}."

score\_Review\_prompt = f"{scores} {Reviews} "

dimension\_prompt = ""

print("dimension",dimension)

if dimension == "Realistic":

dimension\_prompt = (

"Assess the student's artwork based on the 'Realistic' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Realistic. This criterion assesses the accuracy of proportions, textures, lighting, and perspective to create a lifelike depiction."

"\n5: The artwork exhibits exceptional detail and precision in depicting realistic features. Textures and lighting are used masterfully to mimic real-life appearances with accurate proportions and perspective. The representation is strikingly lifelike, demonstrating advanced skills in realism."

"\n4: The artwork presents a high level of detail and accuracy in the portrayal of subjects. Proportions and textures are very well executed, and the lighting enhances the realism. Although highly realistic, minor discrepancies in perspective or detail might be noticeable."

"\n3: The artwork represents subjects with a moderate level of realism. Basic proportions are correct, and some textures and lighting effects are used to enhance realism. However, the depiction may lack depth or detail in certain areas."

"\n2: The artwork attempts realism but struggles with accurate proportions and detailed textures. Lighting and perspective may be inconsistently applied, resulting in a less convincing depiction."

"\n1: The artwork shows minimal attention to realistic details. Proportions, textures, and lighting are poorly executed, making the depiction far from lifelike."

)

elif dimension == "Deformation":

dimension\_prompt = (

"Assess the student's artwork based on the 'Deformation' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Deformation. This criterion evaluates the artist's ability to creatively and intentionally deform reality to convey a message, emotion, or concept."

"\n5: The artwork demonstrates masterful use of deformation to enhance the emotional or conceptual impact of the piece. The transformations are thoughtful and integral to the artwork's message, seamlessly blending with the composition to engage viewers profoundly."

"\n4: The artwork effectively uses deformation to express artistic intentions. The modifications are well-integrated and contribute significantly to the viewer's understanding or emotional response. Minor elements of the deformation might detract from its overall effectiveness."

"\n3: The artwork includes noticeable deformations that add to its artistic expression. While these elements generally support the artwork's theme, they may be somewhat disjointed from the composition, offering mixed impact on the viewer."

"\n2: The artwork attempts to use deformation but does so with limited success. The deformations are present but feel forced or superficial, only marginally contributing to the artwork's expressive goals."

"\n1: The artwork features minimal or ineffective deformation, with little to no enhancement of the artwork's message or emotional impact. The attempts at deformation seem disconnected from the artwork's overall intent."

)

elif dimension == "Imagination":

dimension\_prompt = (

"Assess the student's artwork based on the 'Imagination' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Imagination. This criterion evaluates the artist's ability to use their creativity to form unique and original ideas within their artwork."

"\n5: The artwork displays a profound level of originality and creativity, introducing unique concepts or interpretations that are both surprising and thought-provoking."

"\n4: The artwork presents creative ideas that are both original and nicely executed, though they may be similar to conventional themes."

"\n3: The artwork shows some creative ideas, but they are somewhat predictable and do not stray far from traditional approaches."

"\n2: The artwork has minimal creative elements, with ideas that are largely derivative and lack originality."

"\n1: The artwork lacks imagination, with no discernible original ideas or creative concepts."

)

elif dimension == "Color Richness":

dimension\_prompt = (

"Assess the student's artwork based on the 'Color Richness' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Color Richness. This criterion assesses the use and range of colors to create a visually engaging experience."

"\n5: The artwork uses a wide and harmonious range of colors, each contributing to a vivid and dynamic composition."

"\n4: The artwork features a good variety of colors that are well-balanced, enhancing the visual appeal of the piece."

"\n3: The artwork includes a moderate range of colors, but the palette may not fully enhance the subject matter."

"\n2: The artwork has limited color variety, with a palette that does not significantly contribute to the piece's impact."

"\n1: The artwork shows poor use of colors, with a very restricted range that detracts from the visual experience."

)

elif dimension == "Color Contrast":

dimension\_prompt = (

"Assess the student's artwork based on the 'Color Contrast' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Color Contrast. This criterion evaluates the effective use of contrasting colors to enhance artistic expression."

"\n5: The artwork masterfully employs contrasting colors to create a striking and effective visual impact."

"\n4: The artwork effectively uses contrasting colors to enhance visual interest, though the contrast may be less pronounced."

"\n3: The artwork has some contrast in colors, but it is not used effectively to enhance the artwork's overall appeal."

"\n2: The artwork makes minimal use of color contrast, resulting in a lackluster visual impact."

"\n1: The artwork lacks effective color contrast, making the piece visually unengaging."

)

elif dimension == "Line Combination":

dimension\_prompt = (

"Assess the student's artwork based on the 'Line Combination' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Line Combination. This criterion assesses the integration and interaction of lines within the artwork."

"\n5: The artwork exhibits exceptional integration of line combinations, creating a harmonious and engaging visual flow."

"\n4: The artwork displays good use of line combinations that contribute to the overall composition, though some areas may lack cohesion."

"\n3: The artwork shows average use of line combinations, with some effective sections but overall lacking in cohesiveness."

"\n2: The artwork has minimal effective use of line combinations, with lines that often clash or do not contribute to a unified composition."

"\n1: The artwork shows poor integration of lines, with combinations that disrupt the visual harmony of the piece."

)

elif dimension == "Line Texture":

dimension\_prompt = (

"Assess the student's artwork based on the 'Line Texture' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Line Texture. This criterion evaluates the variety and execution of line textures within the artwork."

"\n5: The artwork demonstrates a wide variety of line textures, each skillfully executed to enhance the piece's aesthetic and thematic elements."

"\n4: The artwork includes a good range of line textures, well executed but with some areas that may lack definition."

"\n3: The artwork features moderate variety in line textures, with generally adequate execution but lacking in detail."

"\n2: The artwork has limited line textures, with execution that does not significantly contribute to the artwork's quality."

"\n1: The artwork lacks variety and sophistication in line textures, resulting in a visually dull piece."

)

elif dimension == "Picture Organization":

dimension\_prompt = (

"Assess the student's artwork based on the 'Picture Organization' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Picture Organization. This criterion evaluates the overall composition and spatial arrangement within the artwork."

"\n5: The artwork is impeccably organized, with each element thoughtfully placed to create a balanced and compelling composition."

"\n4: The artwork has a good organization, with a well-arranged composition that effectively guides the viewer's eye, though minor elements may disrupt the flow."

"\n3: The artwork has an adequate organization, but the composition may feel somewhat unbalanced or disjointed."

"\n2: The artwork shows poor organization, with a composition that lacks coherence and does not effectively engage the viewer."

"\n1: The artwork is poorly organized, with a chaotic composition that detracts from the piece's overall impact."

)

elif dimension == "Transformation":

dimension\_prompt = (

"Assess the student's artwork based on the 'Transformation' criterion. Provide a score (1-5) and a detailed Review explaining the assessment."

"You may receive some Reviews, score feedback from the human user. please consider it more."

"\nCriterion: Transformation. This criterion assesses the artist's ability to transform traditional or familiar elements into something new and unexpected."

"\n5: The artwork is transformative, offering a fresh and innovative take on traditional elements, significantly enhancing the viewer's experience."

"\n4: The artwork successfully transforms familiar elements, providing a new perspective, though the innovation may not be striking."

"\n3: The artwork shows some transformation of familiar elements, but the changes are somewhat predictable and not highly innovative."

"\n2: The artwork attempts transformation but achieves only minimal success, with changes that are either too subtle or not effectively executed."

"\n1: The artwork lacks transformation, with traditional elements that are replicated without any significant innovation or creative reinterpretation."

)

full\_prompt = f"{Entities\_prompt}{score\_Review\_prompt}{dimension\_prompt}\n output a dictionary:"+"{"f" dimension:{dimension},score: from 1 to 5,Review:Review for artworks"+" }"

# 调试语句：打印生成的 prompt 和其长度

print(f"Generated prompt for {dimension}:")

print(full\_prompt)

print(f"Prompt length (in characters): {len(full\_prompt)}")

return full\_prompt

def extract\_score\_Review(response\_content):

data = json.loads(response\_content)

# 提取所需的字段

dimension = data['dimension']

score = data['score']

Review = data['Review']

return score, Review

def extract\_suggestion(response\_content):

data = json.loads(response\_content)

# 提取所需的字段

suggestion = data['suggestion']

return suggestion

def Review\_Generation\_Agent(image\_data, labels\_data, score\_Review\_data,dimension):

prompt = create\_Review\_prompt(labels\_data, score\_Review\_data, dimension)

print("########Review\_Generation\_Agent#######")

print(prompt)

response = client.chat.completions.create(

model="gpt-4o",

temperature=0,

top\_p=1,

messages=[

{"role": "system", "content": prompt

},

{

"role": "user",

"content": [

{"type": "image\_url",

"image\_url": {

"url": f"data:image/jpeg;base64,{image\_data}",

"detail": "high"

},

},

],

}

],

max\_tokens=500,

)

response\_content = response.choices[0].message.content

print(response\_content)

score, Review= extract\_score\_Review(response\_content)

print("score",score)

print("Review",Review)

return score, Review

def Suggestion\_Generation\_Agent(image\_data, labels\_data, score\_Review\_data, suggestion\_data,dimension):

prompt = create\_Suggestion\_prompt(labels\_data, score\_Review\_data, suggestion\_data, dimension)

response = client.chat.completions.create(

model="gpt-4o",

temperature=0,

top\_p=1,

messages=[

{"role": "system", "content": prompt

},

{

"role": "user",

"content": [

{"type": "image\_url",

"image\_url": {

"url": f"data:image/jpeg;base64,{image\_data}",

"detail": "high"

},

},

],

}

],

max\_tokens=500,

)

response\_content = response.choices[0].message.content

print(response\_content)

suggestion = extract\_suggestion(response\_content)

print("suggestion",suggestion)

return suggestion

@app.route('/', methods=['GET', 'POST'])

def upload\_file():

if request.method == 'POST':

if 'file' not in request.files:

flash('No file part')

return redirect(request.url)

file = request.files['file']

if file.filename == '':

flash('No selected file')

return redirect(request.url)

if file and allowed\_file(file.filename):

filename = secure\_filename(file.filename)

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(filepath)

base64\_image = encode\_image(filepath)

try:

# 重置全局评分数据

for key in global\_scores:

global\_scores[key] = 0

# 检查是否已有标签文件

labels\_filepath = os.path.join('userActions/Entities', f"{filename}\_labels.json")

if os.path.exists(labels\_filepath):

with open(labels\_filepath, 'r') as f:

labels\_data = json.load(f)

identified\_objects = labels\_data['original']

style\_data = labels\_data['style']

else:

# 识别图像中的物体并生成标签

identified\_objects, style\_label = Entity\_Recognition\_Agent(base64\_image)

# 初始化标签和风格数据

labels\_data = {"original": identified\_objects, "added": [], "removed": []}

print("style\_label", style\_label)

style\_data = {"original": [style\_label], "added": [], "removed": []}

print("style\_data",style\_data)

# 保存标签和风格到JSON文件

data\_to\_save = {"original": labels\_data["original"], "added": [], "removed": [], "style": style\_data}

with open(labels\_filepath, 'w') as f:

json.dump(data\_to\_save, f)

print(f"Successfully saved labels and style to file: {labels\_filepath}")

# 初始化评分为0

initial\_scores = [0] \* 9 # 假设有9个维度

radar\_chart = plot\_radar\_chart(initial\_scores)

return render\_template('result.html',

image\_data=base64\_image,

identified\_objects=identified\_objects,

style\_label=style\_data['original'], # 传递艺术风格标签

radar\_chart=radar\_chart,

image\_name=filename # 传递文件名

)

except Exception as e:

print(e)

flash(f"An unexpected error occurred: {str(e)}")

return redirect(request.url)

return render\_template('upload.html')

@app.route('/evaluate\_dimension', methods=['POST'])

def evaluate\_dimension():

data = request.json

image\_data = data['image\_data']

dimension = data['dimension']

image\_name = data['image\_name']

# 读取标签信息

labels\_filepath = os.path.join('userActions/Entities', f"{image\_name}\_labels.json")

if os.path.exists(labels\_filepath):

with open(labels\_filepath, 'r') as f:

labels\_data = json.load(f)

else:

labels\_data = None

# 读取评分和评论信息

score\_Review\_filepath = os.path.join('userActions/score\_Review', f"{image\_name}\_{dimension}\_score\_Review.json")

if os.path.exists(score\_Review\_filepath):

with open(score\_Review\_filepath, 'r') as f:

score\_Review\_data = json.load(f)

else:

score\_Review\_data = {

"scores": {"original": 0, "current": 0, "initGPTscore": None},

"Reviews": {"original": "", "current": "", "added": "", "removed": ""},

}

# 确保 'initGPTscore' 键存在并被初始化

if 'initGPTscore' not in score\_Review\_data['scores']:

score\_Review\_data['scores']['initGPTscore'] = None

# 保存当前 round 的数据

save\_round\_data(image\_name, dimension, 'score\_Review', score\_Review\_data)

score, Review = Review\_Generation\_Agent(image\_data, labels\_data, score\_Review\_data, dimension)

# 如果 initGPTscore 尚未设置，则将其设置为初始得分

if score\_Review\_data['scores']['initGPTscore'] is None:

score\_Review\_data['scores']['initGPTscore'] = score

# 更新并覆盖原始评论和评分

score\_Review\_data['scores']['original'] = score

score\_Review\_data['scores']['current'] = score

score\_Review\_data['Reviews']['original'] = Review

score\_Review\_data['Reviews']['current'] = Review

score\_Review\_data['Reviews']['added'] = ""

score\_Review\_data['Reviews']['removed'] = ""

# 覆盖保存新的评分和评论

with open(score\_Review\_filepath, 'w') as f:

json.dump(score\_Review\_data, f)

# 更新全局评分数据

if score is not None:

global\_scores[dimension] = score

# 打印最终返回的数据

print(f"Evaluating {dimension}: score={score}, Review={Review}")

return jsonify({

"score": score,

"Review": Review,

"initGPTscore": score\_Review\_data['scores']['initGPTscore']

})

@app.route('/generate\_suggestion', methods=['POST'])

def generate\_suggestion():

data = request.json

image\_data = data['image\_data']

dimension = data['dimension']

image\_name = data['image\_name']

try:

# 读取标签信息

try:

labels\_filepath = os.path.join('userActions/Entities', f"{image\_name}\_labels.json")

if os.path.exists(labels\_filepath):

with open(labels\_filepath, 'r') as f:

labels\_data = json.load(f)

else:

labels\_data = None

except Exception as e:

print(f"Error loading labels data: {e}")

labels\_data = None

# 读取评分和评论信息

try:

score\_Review\_filepath = os.path.join('userActions/score\_Review',

f"{image\_name}\_{dimension}\_score\_Review.json")

if os.path.exists(score\_Review\_filepath):

with open(score\_Review\_filepath, 'r') as f:

score\_Review\_data = json.load(f)

else:

score\_Review\_data = None

except Exception as e:

print(f"Error loading score/Review data: {e}")

score\_Review\_data = None

# 读取建议信息

try:

suggestion\_filepath = os.path.join('userActions/suggestion', f"{image\_name}\_{dimension}\_suggestion.json")

if os.path.exists(suggestion\_filepath):

with open(suggestion\_filepath, 'r') as f:

suggestion\_data = json.load(f)

else:

suggestion\_data = {"suggestions": {"original": "", "current": "", "added": "", "removed": ""}}

# 确保 suggestion\_data 结构完整

if 'suggestions' not in suggestion\_data:

suggestion\_data = {"suggestions": {"original": "", "current": "", "added": "", "removed": ""}}

except Exception as e:

print(f"Error loading suggestion data: {e}")

suggestion\_data = {"suggestions": {"original": "", "current": "", "added": "", "removed": ""}}

# 保存现有的建议数据到 everyround 数据文件中（第一件事）

try:

save\_round\_data(image\_name, dimension, 'suggestion', suggestion\_data)

except Exception as e:

print(f"Error saving round data (before generation): {e}")

suggestion = Suggestion\_Generation\_Agent(image\_data, labels\_data, score\_Review\_data, suggestion\_data, dimension)

try:

suggestion\_data = {

"suggestions": {

"original": suggestion,

"current": suggestion,

"added": "",

"removed": ""

}

}

# 覆盖保存生成的建议

with open(suggestion\_filepath, 'w') as f:

json.dump(suggestion\_data, f)

print(f"Successfully saved suggestion file: {suggestion\_filepath}")

except Exception as e:

print(f"Error saving generated suggestion data: {e}")

return jsonify({"error": "An error occurred during suggestion saving."}), 500

return jsonify({"suggestion": suggestion})

except Exception as e:

print(f"An unexpected error occurred: {e}")

return jsonify({"error": "An unexpected error occurred, please try again later."}), 500

@app.route('/update\_radar\_chart', methods=['POST'])

def update\_radar\_chart():

scores = list(global\_scores.values()) # 获取当前所有评分

radar\_chart = plot\_radar\_chart(scores)

return jsonify({"radar\_chart": radar\_chart})

def process\_text\_change(old\_text, new\_text):

"""处理文本的变化，返回 added 和 removed 的内容，忽略中间未修改的部分"""

removed = []

added = []

# 使用 difflib.SequenceMatcher 来找出精确的变化部分

matcher = difflib.SequenceMatcher(None, old\_text, new\_text)

# 遍历操作码，识别哪些部分是替换、删除或添加的

for tag, i1, i2, j1, j2 in matcher.get\_opcodes():

if tag == 'replace':

# 替换操作 - 在 old\_text 中的部分被替换为 new\_text 中的部分

removed.append(old\_text[i1:i2])

added.append(new\_text[j1:j2])

elif tag == 'delete':

# 删除操作 - old\_text 中的部分被删除

removed.append(old\_text[i1:i2])

elif tag == 'insert':

# 插入操作 - 在 new\_text 中的部分被添加

added.append(new\_text[j1:j2])

# 将结果连接成字符串

return ''.join(added), ''.join(removed)

@app.route('/save\_user\_actions', methods=['POST'])

def save\_user\_actions():

data = request.json

image\_name = data.get('image\_name', 'default\_image')

user\_actions = data.get('user\_actions', {})

print('user\_actions',user\_actions)

scores = user\_actions.get('scores', {})

Reviews = user\_actions.get('Reviews', {})

suggestions = user\_actions.get('suggestions', {}) # 获取 suggestions 数据

Entities = user\_actions.get('Entities', {})

print("##save##")

print("Entities",Entities)

style = user\_actions.get('style', {}) # 获取 style 数据

print("Style",style)

# 定义三个文件夹

score\_Review\_folder = 'userActions/score\_Review'

suggestion\_folder = 'userActions/suggestion'

Entities\_folder = 'userActions/Entities'

# 创建文件夹

os.makedirs(score\_Review\_folder, exist\_ok=True)

os.makedirs(suggestion\_folder, exist\_ok=True)

os.makedirs(Entities\_folder, exist\_ok=True)

# 保存评分和评论数据

if scores or Reviews:

for dimension in scores.keys():

score\_Review\_filename = f"{image\_name}\_{dimension}\_score\_Review.json"

score\_Review\_filepath = os.path.join(score\_Review\_folder, score\_Review\_filename)

if os.path.exists(score\_Review\_filepath):

with open(score\_Review\_filepath, 'r') as f:

existing\_data = json.load(f)

else:

existing\_data = {

"scores": {"original": 0, "current": 0, "initGPTscore": None},

"Reviews": {"original": "", "current": "", "added": "", "removed": ""},

}

new\_score = scores.get(dimension, {}).get('current', existing\_data['scores']['current'])

new\_Review = Reviews.get(dimension, {}).get('current', existing\_data['Reviews']['current'])

# 更新 original, current, added, removed

Review\_added, Review\_removed = process\_text\_change(existing\_data['Reviews']['original'], new\_Review)

existing\_data['Reviews']['added'] = Review\_added

existing\_data['Reviews']['removed'] = Review\_removed

existing\_data['scores']['current'] = new\_score

existing\_data['Reviews']['current'] = new\_Review

if existing\_data['scores']['original'] == 0:

existing\_data['scores']['original'] = new\_score

if existing\_data['Reviews']['original'] == "":

existing\_data['Reviews']['original'] = new\_Review

# 确保 initGPTscore 保持不变

if existing\_data['scores']['initGPTscore'] is None:

existing\_data['scores']['initGPTscore'] = new\_score

try:

with open(score\_Review\_filepath, 'w') as f:

json.dump(existing\_data, f)

print(f"Successfully saved score and Review file: {score\_Review\_filepath}")

except Exception as e:

print(f"Failed to save score and Review file: {score\_Review\_filepath}, Error: {e}")

# 保存建议数据

if suggestions:

for dimension in suggestions.keys():

suggestion\_filename = f"{image\_name}\_{dimension}\_suggestion.json"

suggestion\_filepath = os.path.join(suggestion\_folder, suggestion\_filename)

if os.path.exists(suggestion\_filepath):

with open(suggestion\_filepath, 'r') as f:

existing\_data = json.load(f)

else:

existing\_data = {"suggestions": {"original": "", "current": "", "added": "", "removed": ""}}

new\_suggestion = suggestions.get(dimension, {}).get('current', existing\_data['suggestions']['current'])

# 更新 original, current, added, removed

suggestion\_added, suggestion\_removed = process\_text\_change(existing\_data['suggestions']['original'],

new\_suggestion)

existing\_data['suggestions']['current'] = new\_suggestion

existing\_data['suggestions']['added'] = suggestion\_added

existing\_data['suggestions']['removed'] = suggestion\_removed

if existing\_data['suggestions']['original'] == "":

existing\_data['suggestions']['original'] = new\_suggestion

try:

with open(suggestion\_filepath, 'w') as f:

json.dump(existing\_data, f)

print(f"Successfully saved suggestion file: {suggestion\_filepath}")

except Exception as e:

print(f"Failed to save suggestion file: {suggestion\_filepath}, Error: {e}")

# 保存标签数据，包括风格标签

if Entities.get('added') or Entities.get('removed') or style.get('added') or style.get('removed'):

labels\_filename = f"{image\_name}\_labels.json"

labels\_filepath = os.path.join(Entities\_folder, labels\_filename)

try:

# 加载现有的标签和风格数据，确保 style 不被修改

if os.path.exists(labels\_filepath):

with open(labels\_filepath, 'r') as f:

existing\_data = json.load(f)

style\_data = existing\_data.get("style", {"original": [], "added": [], "removed": []})

else:

style\_data = {"original": [], "added": [], "removed": []}

# 更新普通标签

data\_to\_save = {"original": Entities.get('original', []), "added": Entities.get('added', []), "removed": Entities.get('removed', []), "style": style}

# 保存更新后的数据

with open(labels\_filepath, 'w') as f:

json.dump(data\_to\_save, f)

print(f"Successfully saved labels and style to file: {labels\_filepath}")

except Exception as e:

print(f"Failed to save labels and style to file: {labels\_filepath}, Error: {e}")

return jsonify({"status": "success"})

@app.route('/submit\_score\_Review', methods=['POST'])

def submit\_score\_Review():

data = request.json

image\_name = data['image\_name']

dimension = data['dimension']

# 检查dimension是否被正确传递

print(f"Dimension received: {dimension}")

try:

score\_Review\_filepath = os.path.join('userActions/score\_Review',

f"{image\_name}\_{dimension}\_score\_Review.json")

if os.path.exists(score\_Review\_filepath):

with open(score\_Review\_filepath, 'r') as f:

score\_Review\_data = json.load(f)

else:

score\_Review\_data = {

"scores": {"original": 0, "current": 0, "initGPTscore": None},

"Reviews": {"original": "", "current": "", "added": "", "removed": ""},

}

except Exception as e:

print(f"Error loading score/Review data: {e}")

score\_Review\_data = {

"scores": {"original": 0, "current": 0, "initGPTscore": None},

"Reviews": {"original": "", "current": "", "added": "", "removed": ""},

}

save\_round\_data(image\_name, dimension, 'score\_Review', score\_Review\_data)

return jsonify({"status": "success", "message": f"Score and Review for {dimension} submitted successfully"})

@app.route('/submit\_suggestion', methods=['POST'])

def submit\_suggestion():

data = request.json

image\_name = data['image\_name']

dimension = data['dimension']

# 检查dimension是否被正确传递

print(f"Dimension received: {dimension}")

try:

suggestion\_filepath = os.path.join('userActions/suggestion',

f"{image\_name}\_{dimension}\_suggestion.json")

if os.path.exists(suggestion\_filepath):

with open(suggestion\_filepath, 'r') as f:

suggestion\_data = json.load(f)

else:

suggestion\_data = {"suggestions": {"original": "","current": "","added": "","removed": ""}}

except Exception as e:

print(f"Error loading score/Review data: {e}")

suggestion\_data = {"suggestions": {"original": "","current": "","added": "","removed": ""}}

# 调用 save\_round\_data 函数保存最后一轮的 suggestion 数据

save\_round\_data(image\_name, dimension, 'suggestion', suggestion\_data)

return jsonify({"status": "success", "message": f"Suggestion for {dimension} submitted successfully"})

# 保存每轮的评分和建议数据，并独立计算round

def save\_round\_data(image\_name, dimension, data\_type, new\_data):

# 定义不同的文件夹

folder = os.path.join('userActionsEveryRounds', data\_type)

os.makedirs(folder, exist\_ok=True)

# 初始化或增加 round 计数器，确保score\_Review和suggestion独立计算

if image\_name not in round\_counters:

round\_counters[image\_name] = {'score\_Review': {}, 'suggestion': {}}

# 确保score\_Review和suggestion的轮次是独立的

round\_counters[image\_name][data\_type].setdefault(dimension, 0)

round\_counters[image\_name][data\_type][dimension] += 1

current\_round = round\_counters[image\_name][data\_type][dimension]

print(f"Current round for {dimension} ({data\_type}): {current\_round}")

# 构建文件名

round\_filename = f"{image\_name}\_{dimension}\_{data\_type}.json"

round\_filepath = os.path.join(folder, round\_filename)

print(f"Saving round data for {dimension} ({data\_type}) at {round\_filepath}")

# 添加调试信息，打印保存数据的路径和内容

print(f"Saving round data to {round\_filepath}")

print(f"Current round: {current\_round}")

print(f"Data to be saved: {new\_data}")

# 读取之前的数据并合并

if os.path.exists(round\_filepath):

with open(round\_filepath, 'r') as f:

previous\_rounds\_data = json.load(f)

else:

previous\_rounds\_data = []

print(f"Previous rounds data for {dimension}: {previous\_rounds\_data}")

# 添加当前轮次的新数据

previous\_rounds\_data.append({'round': current\_round, 'data': new\_data})

# 保存合并后的数据到文件

with open(round\_filepath, 'w') as f:

json.dump(previous\_rounds\_data, f)

print(f"Successfully saved round data for {dimension} in round {current\_round}")

# 画雷达图

def plot\_radar\_chart(scores):

labels = np.array([

'Realistic', 'Deformation', 'Imagination', 'Color Richness', 'Color Contrast',

'Line Combination', 'Line Texture', 'Picture Organization', 'Transformation'

])

num\_vars = len(labels)

# Compute angle for each axis

angles = np.linspace(0, 2 \* np.pi, num\_vars, endpoint=False).tolist()

# The radar chart is a circle, so it needs to be "closed".

scores += scores[:1]

angles += angles[:1]

fig, ax = plt.subplots(figsize=(4, 3), subplot\_kw=dict(polar=True))

ax.fill(angles, scores, color='red', alpha=0.25)

ax.plot(angles, scores, color='red', linewidth=2)

plt.xticks(angles[:-1], labels, fontsize=12) # 设置坐标轴字体大小

# Set the range of the radar chart

ax.set\_ylim(0, 5)

ax.set\_yticks([1, 2, 3, 4, 5])

ax.set\_yticklabels(['1', '2', '3', '4', '5'], fontsize=12) # 设置刻度标签的字体大小

# Save the plot as a PNG image with higher DPI for better clarity

buf = BytesIO()

plt.savefig(buf, format='png', dpi=300) # You can adjust DPI based on your requirement

base64\_image = base64.b64encode(buf.getvalue()).decode('utf-8')

buf.close()

plt.close(fig) # Close the figure to free memory

return base64\_image

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

import os

import json

def process\_json\_files(file\_path):

if not os.path.exists(file\_path):

print(f"File {file\_path} not found, skipping.")

with open(file\_path, 'r', encoding='utf-8') as file:

json\_data = json.load(file)

E = json\_data.get("original", [])

R = json\_data.get("added", [])

W = json\_data.get("removed", [])

TP = len(E) - len(W)

MR = min(len(W), len(R))

FP = max(0, len(W) - MR)

FN = max(0, len(R) - MR)

return TP, FP, FN, MR

def get\_Entity\_Accuracy(TP, FP, FN, MR):

Accuracy = TP / (TP + FP + FN + MR) if (TP + FP + FN + MR) > 0 else 0

return Accuracy

def get\_Entity\_Precision(TP, FP, MR):

Precision = TP / (TP + FP + MR) if (TP + FP + MR) > 0 else 0

return Precision

def get\_Entity\_Recall(TP, FN, MR):

Recall = TP / (TP + FN + MR) if (TP + FN + MR) > 0 else 0

return Recall

def get\_Entity\_F1(Precision, Recall):

F1 = 2 \* (Precision \* Recall) / (Precision + Recall) if (Precision + Recall) > 0 else 0

return F1

import os

import json

import numpy as np

import pandas as pd

# Dimension list

dimensions = [

"Realistic", "Deformation", "Imagination", "Color Richness",

"Color Contrast", "Line Combination", "Line Texture",

"Picture Organization", "Transformation"

]

# Load JSON file function

def load\_json\_data(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return json.load(f)

except FileNotFoundError:

print(f"File not found: {file\_path}")

return None

# Calculate Score Volatility (SV)

def calculate\_sv(score\_data):

user\_scores = []

for round\_data in score\_data:

if round\_data["round"] == 1:

continue # Skip round 1 initialization data

# Get user score

user\_score = round\_data['data']['scores']['current']

if user\_score is not None:

user\_scores.append(float(user\_score))

if len(user\_scores) < 2:

return np.nan # Return NaN if not enough data

# Calculate standard deviation to measure score volatility

sv = np.std(user\_scores)

return sv

# Process each image and each dimension to calculate SV

def process\_directory\_for\_sv(score\_review\_dir, output\_file\_sv):

sv\_results = []

for image\_num in range(1, 21): # Adjust according to the actual number of images

sv\_values = []

print(f"Processing image number: {image\_num}")

for dimension in dimensions:

# Load score\_Review file

score\_review\_file = os.path.join(score\_review\_dir, f"{image\_num}.jpg\_{dimension}\_score\_Review.json")

score\_review\_data = load\_json\_data(score\_review\_file)

if score\_review\_data:

print(f"Processing file: {score\_review\_file}")

# Calculate score volatility

sv\_value = calculate\_sv(score\_review\_data)

sv\_values.append(sv\_value)

# Calculate average SV for each image across dimensions

avg\_sv = np.nanmean(sv\_values)

print(f"Image {image\_num}'s average SV: {avg\_sv}")

# Save results

sv\_results.append([f"{image\_num}.jpg", avg\_sv])

# Save results to DataFrame and output to Excel file

sv\_df = pd.DataFrame(sv\_results, columns=["File Name", "SV"])

sv\_df.to\_excel(output\_file\_sv, index=False)

print(f"SV results have been saved to: {output\_file\_sv}")

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

score\_review\_directory = "userActionsEveryRounds/score\_Review" # JSON file path

output\_file\_sv = "SV\_Results.xlsx"

process\_directory\_for\_sv(score\_review\_directory, output\_file\_sv)

import os

import json

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import Normalize

from matplotlib.patches import FancyBboxPatch

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from matplotlib.cm import ScalarMappable

import pandas as pd # 引入 pandas 库

# 自定义颜色映射，渐变从灰白色到深绿色

cmap = plt.cm.get\_cmap("Greens") # 使用 Matplotlib 的 "Greens" 渐变色

gray\_color = "#0f0f0f" # 用于表示无数据的灰白色

norm = Normalize(vmin=0, vmax=1)

# 定义9个维度名称

dimensions = [

"Realistic", "Deformation", "Imagination", "Color Richness",

"Color Contrast", "Line Combination", "Line Texture",

"Picture Organization", "Transformation"

]

# 1. 加载JSON数据

def load\_json\_data(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return json.load(f)

except FileNotFoundError:

print(f"File not found: {file\_path}")

return None

# 2. 定义标准化函数

def normalize(value, min\_value, max\_value):

if max\_value - min\_value == 0:

return 0

return (value - min\_value) / (max\_value - min\_value)

# 3. 计算评分差异度（SD），并反转其值，让差异越小越接近1

def calculate\_sd(score\_data):

total\_diff = 0

valid\_rounds = 0

for round\_data in score\_data:

if round\_data["round"] == 1:

continue # 跳过初始化的 round 1 数据

gpt\_score = round\_data['data']['scores']['initGPTscore']

user\_score = round\_data['data']['scores']['current']

if gpt\_score is not None and user\_score is not None:

try:

gpt\_score = float(gpt\_score)

user\_score = float(user\_score)

score\_diff = abs(gpt\_score - user\_score)

total\_diff += score\_diff

valid\_rounds += 1

except ValueError:

continue

if valid\_rounds == 0:

return np.nan # 无数据则返回空值

sd = total\_diff / valid\_rounds

normalized\_sd = 1 - normalize(sd, 0, 5)

return normalized\_sd

# 4. 计算SC (评分接受度)

def calculate\_sc(score\_data):

gpt\_scores = []

user\_scores = []

for round\_data in score\_data:

if round\_data["round"] == 1:

continue

gpt\_score = round\_data['data']['scores']['initGPTscore']

user\_score = round\_data['data']['scores']['current']

if gpt\_score is not None and user\_score is not None:

gpt\_scores.append(float(gpt\_score))

user\_scores.append(float(user\_score))

if len(gpt\_scores) < 1 or len(user\_scores) < 1:

return np.nan # 无数据则返回空值

score\_diffs = np.abs(np.array(gpt\_scores) - np.array(user\_scores))

max\_diff = 5.0

normalized\_diff = 1 - (np.mean(score\_diffs) / max\_diff)

return normalize(normalized\_diff, 0, 1)

# 5. 计算SV（评分波动度），并反转其值

def calculate\_sv(score\_data):

user\_scores = []

for round\_data in score\_data:

if round\_data["round"] == 1:

continue

user\_score = round\_data['data']['scores']['current']

if user\_score is not None:

user\_scores.append(float(user\_score))

if len(user\_scores) <= 0:

return np.nan # 无数据则返回空值

sv = np.std(user\_scores)

normalized\_sv = 1 - normalize(sv, 0, 5)

return normalized\_sv

# 6. 计算TAR（文本修改率），并反转其值

def calculate\_tar(text\_data):

total\_added = total\_removed = total\_gpt\_length = 0

for round\_data in text\_data:

if round\_data["round"] == 1:

continue

added = round\_data['data']['Reviews']['added']

removed = round\_data['data']['Reviews']['removed']

original = round\_data['data']['Reviews']['original']

total\_added += len(added)

total\_removed += len(removed)

total\_gpt\_length += len(original)

if total\_added + total\_gpt\_length > 0:

tar = total\_removed / (total\_added + total\_gpt\_length)

normalized\_tar = 1 - normalize(tar, 0, 1)

return normalized\_tar

return np.nan # 无数据则返回空值

# 7. 计算文本相似度（TS）

def calculate\_text\_similarity(text\_data, is\_suggestion=False):

gpt\_texts = []

user\_texts = []

for round\_data in text\_data:

if round\_data["round"] == 1:

continue

if is\_suggestion:

gpt\_text = round\_data['data']['suggestions']['original']

user\_text = round\_data['data']['suggestions']['current']

else:

gpt\_text = round\_data['data']['Reviews']['original']

user\_text = round\_data['data']['Reviews']['current']

if gpt\_text.strip() and user\_text.strip():

gpt\_texts.append(gpt\_text)

user\_texts.append(user\_text)

if not gpt\_texts or not user\_texts:

return np.nan # 无数据则返回空值

vectorizer = CountVectorizer(analyzer='char').fit\_transform([gpt\_texts[-1], user\_texts[-1]])

vectors = vectorizer.toarray()

if len(vectors) > 1:

similarity = cosine\_similarity(vectors)[0][1]

return normalize(similarity, 0, 1)

return np.nan # 无数据则返回空值

# 生成圆角矩形

def draw\_rounded\_square(ax, color, x, y, size=0.9, radius=0.2):

square = FancyBboxPatch((x, y), size, size,

boxstyle=f"round,pad=0,rounding\_size={radius}",

facecolor=color, edgecolor='none')

ax.add\_patch(square)

# 生成华夫饼图

def plot\_custom\_waffle\_chart(image\_metrics, metric\_name):

n\_rows = 4 # 固定每行4个格子

n\_cols = 5 # 固定每列5个格子

values = [metrics[metric\_name] for metrics in image\_metrics]

fig, ax = plt.subplots(figsize=(10, 4))

ax.set\_xlim(0, n\_cols)

ax.set\_ylim(0, n\_rows)

ax.set\_xticks([])

ax.set\_yticks([])

ax.set\_aspect('equal')

for i, value in enumerate(values):

x = i % n\_cols

y = n\_rows - 1 - i

color = gray\_color if np.isnan(value) else cmap(norm(value))

draw\_rounded\_square(ax, color, x, y)

# 添加颜色渐变图例

sm = ScalarMappable(cmap=cmap, norm=norm)

sm.set\_array([])

cbar = plt.colorbar(sm, ax=ax, orientation='horizontal', fraction=0.03, pad=0.04)

cbar.set\_label(metric\_name)

plt.show()

# 9. 批量处理文件并计算每张图片的指标

def process\_directory(score\_review\_dir, suggestion\_dir):

image\_metrics = []

for image\_num in range(1, 21):

sc\_values, sv\_values, tar\_values, ts\_values, sd\_values = [], [], [], [], []

for dimension in dimensions:

score\_review\_file = os.path.join(score\_review\_dir, f"{image\_num}.jpg\_{dimension}\_score\_Review.json")

suggestion\_file = os.path.join(suggestion\_dir, f"{image\_num}.jpg\_{dimension}\_suggestion.json")

score\_review\_data = load\_json\_data(score\_review\_file)

if score\_review\_data:

sc\_values.append(calculate\_sc(score\_review\_data))

sv\_values.append(calculate\_sv(score\_review\_data))

tar\_values.append(calculate\_tar(score\_review\_data))

sd\_values.append(calculate\_sd(score\_review\_data))

ts\_values.append(calculate\_text\_similarity(score\_review\_data))

suggestion\_data = load\_json\_data(suggestion\_file)

if suggestion\_data:

ts\_values.append(calculate\_text\_similarity(suggestion\_data, is\_suggestion=True))

if sc\_values:

avg\_sc = np.nanmean(sc\_values)

avg\_sv = np.nanmean(sv\_values)

avg\_tar = np.nanmean(tar\_values)

avg\_ts = np.nanmean(ts\_values)

avg\_sd = np.nanmean(sd\_values)

# 打印每张图片的各个指标

print(f"Image: {image\_num}.jpg, SC: {avg\_sc}, SV: {avg\_sv}, TAR: {avg\_tar}, TS: {avg\_ts}, SD: {avg\_sd}")

image\_metrics.append({

"image": f"{image\_num}.jpg",

"SC": avg\_sc,

"SV": avg\_sv,

"TAR": avg\_tar,

"TS": avg\_ts,

"SD": avg\_sd

})

# 将结果保存到 DataFrame

df = pd.DataFrame(image\_metrics)

# 导出到 Excel 文件

output\_file = "image\_metrics.xlsx"

df.to\_excel(output\_file, index=False)

print(f"Metrics exported to {output\_file}")

# 生成各项指标的华夫饼图

plot\_custom\_waffle\_chart(image\_metrics, "SC")

plot\_custom\_waffle\_chart(image\_metrics, "SV")

plot\_custom\_waffle\_chart(image\_metrics, "TAR")

plot\_custom\_waffle\_chart(image\_metrics, "TS")

plot\_custom\_waffle\_chart(image\_metrics, "SD")

# 主函数入口

if \_\_name\_\_ == "\_\_main\_\_":

# 指定存放JSON文件的目录

score\_review\_directory = "score\_Review"

suggestion\_directory = "suggestion"

process\_directory(score\_review\_directory, suggestion\_directory)

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<title>ArtMentor</title>

<style>

body {

font-family: 'Arial', sans-serif;

color: #333;

margin: 0;

padding: 0;

background: radial-gradient(ellipse at center, #fffeea 0%, #fffeea 35%, #b7e8eb 100%);

display: flex;

flex-direction: column;

min-height: 100vh;

justify-content: space-between;

align-items: center;

}

h1 {

font-family: 'Brush Script MT', cursive;

color: #4a4a4a;

margin: 20px 0;

}

.container {

display: flex;

width: 100%;

box-sizing: border-box;

align-items: flex-start;

}

.left-column {

box-sizing: border-box;

display: flex;

flex-direction: column;

align-items: center;

top: 30px;

}

.left-column::-webkit-scrollbar {

display: none;

}

.left-column img {

width: 100%;

height: 300px;

border-radius: 10px;

margin-bottom: 20px;

}

.Entities-section, .radar-section {

margin-top: 20px;

}

.Entities-section h2, .radar-section h2 {

font-family: 'Verdana', sans-serif;

font-weight: bold;

margin-bottom: 10px;

}

#Entities {

margin-top: 10px;

display: flex;

flex-wrap: wrap;

justify-content: center;

}

.Entity {

display: inline-block;

background-color: #FFFBF9;

color: #333333;

padding: 8px 10px;

margin: 8px;

border-radius: 25px;

font-size: 14px;

font-family: "Linux Libertine";

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

}

.Entity .remove {

margin-left: 10px;

color: red;

cursor: pointer;

font-weight: bold;

}

input[type="text"] {

width: 80%;

padding: 12px 15px;

margin-top: 10px;

border-radius: 25px;

border: 1px solid #ddd;

font-size: 14px;

font-family: "Linux Libertine";

background-color: #FFFBF9;

box-shadow: inset 0 2px 4px rgba(0, 0, 0, 0.1);

}

button {

background-color: #F5FCD5;

color: #333333;

padding: 12px 25px;

border-radius: 25px;

border: none;

cursor: pointer;

font-size: 14px;

font-family: "Linux Libertine";

margin-top: 10px;

transition: background-color 0.3s ease;

}

button:hover {

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

background-color: #FFFEF9;

}

#styleEntity{

background-color: gold;

}

.middle-column {

padding: 0 20px;

box-sizing: border-box;

display: flex;

flex-direction: column;

align-items: center;

}

.middle-column::-webkit-scrollbar {

display: none;

}

.dimension-section {

background-color: #fff;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

padding: 20px;

font-family: "Linux Libertine";

width: 110%;

}

.dimension-section h3 {

margin-bottom: 15px;

color: #333;

}

select, textarea {

width: 100%;

padding: 10px 15px;

margin: 10px 0;

border-radius: 25px;

border: 1px solid #ddd;

font-size: 14px;

background-color: #FFFBF9;

box-sizing: border-box;

}

textarea {

height: 100px;

resize: none;

}

select {

appearance: none;

padding-right: 35px;

background-image: url("data:image/svg+xml,%3Csvg xmlns='http:

background-repeat: no-repeat;

font-family: "Linux Libertine";

background-position: right 15px center;

}

.loading {

display: none;

font-size: 18px;

color: #333;

margin-top: 10px;

}

.loading {

display: none;

font-size: 18px;

color: #333;

margin-top: 10px;

}

.loading.show {

display: block;

}

.spinner {

border-radius: 50%;

}

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

.right-column {

display: flex;

flex-direction: column;

align-items: center;

box-sizing: border-box;

top: 80px;

height:100%;

}

.dimension-buttons {

display: flex;

flex-direction: column;

align-items: center;

background-color: #FFFBF9;

padding: 20px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

position: sticky;

top: 20px;

}

.dimension-buttons button {

margin-bottom: 10px;

padding: 10px 15px;

width: 100%;

}

a {

margin: 20px 0;

color: #4a4a4a;

text-decoration: none;

}

a:hover {

text-decoration: underline;

}

.Entities-section, .radar-section {

display: flex;

flex-direction: column;

align-items: center;

}

</style>

</head>

<body>

<h1>ArtMentor</h1>

<div class="container">

<div class="left-column">

<div class="dimension-section">

<img src="data:image/png;base64,{{ image\_data }}" alt="Uploaded Image">

</div>

<div class="dimension-section">

<div class="Entities-section">

<h2>Entity Recognition</h2>

<div id="Entities">

{% for obj in identified\_objects %}

<div class="Entity" id="Entity">{{ obj }}<span class="remove" onclick="removeEntity(this)">x</span></div>

{% endfor %}

{% if style\_label %}

<div class="Entity" id="styleEntity">Style: {{ style\_label[0] }}<span class="remove" onclick="removeEntity(this, true)">x</span></div>

{% endif %}

</div>

<input type="text" id="newEntity" placeholder="Add a New Entity">

<button onclick="addEntity()">Add Entity</button>

</div>

</div>

<div class="dimension-section">

<div class="radar-section">

<h2>Radar</h2>

<img id="radar-chart" src="data:image/png;base64,{{ radar\_chart }}" alt="Radar Chart">

</div>

</div>

</div>

<div class="middle-column">

<h2>Evaluation Sections</h2>

<div id="evaluation-sections">

<div class="dimension-section" id="Realistic-section">

<h3>Realistic</h3>

<button id="evaluate-Realistic-btn" onclick="evaluateDimension('Realistic')">Evaluate Realistic</button>

<button id="generate-Realistic-suggestion-btn" onclick="generateSuggestion('Realistic')">Generate Realistic Suggestion</button>

<button id="submit-Realistic-score-Review-btn" onclick="submitScoreReview('Realistic')">Submit Score & Review</button>

<button id="submit-Realistic-suggestion-btn" onclick="submitSuggestion('Realistic')">Submit Suggestion</button>

<div id="loading-Realistic" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="Realistic-score" onchange="recordScoreChange('Realistic')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="Realistic-Review" oninput="recordReviewChange('Realistic')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="Realistic-suggestion" oninput="recordSuggestionChange('Realistic')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="Deformation-section">

<h3>Deformation</h3>

<button id="evaluate-Deformation-btn" onclick="evaluateDimension('Deformation')">Evaluate Deformation</button>

<button id="generate-Deformation-suggestion-btn" onclick="generateSuggestion('Deformation')">Generate Deformation Suggestion</button>

<button id="submit-Deformation-score-Review-btn" onclick="submitScoreReview('Deformation')">Submit Score & Review</button>

<button id="submit-Deformation-suggestion-btn" onclick="submitSuggestion('Deformation')">Submit Suggestion</button>

<div id="loading-Deformation" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="Deformation-score" onchange="recordScoreChange('Deformation')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="Deformation-Review" oninput="recordReviewChange('Deformation')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="Deformation-suggestion" oninput="recordSuggestionChange('Deformation')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="Imagination-section">

<h3>Imagination</h3>

<button id="evaluate-Imagination-btn" onclick="evaluateDimension('Imagination')">Evaluate Imagination</button>

<button id="generate-Imagination-suggestion-btn" onclick="generateSuggestion('Imagination')">Generate Imagination Suggestion</button>

<button id="submit-Imagination-score-Review-btn" onclick="submitScoreReview('Imagination')">Submit Score & Review</button>

<button id="submit-Imagination-suggestion-btn" onclick="submitSuggestion('Imagination')">Submit Suggestion</button>

<div id="loading-Imagination" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="Imagination-score" onchange="recordScoreChange('Imagination')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="Imagination-Review" oninput="recordReviewChange('Imagination')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="Imagination-suggestion" oninput="recordSuggestionChange('Imagination')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="ColorRichness-section">

<h3>Color Richness</h3>

<button id="evaluate-ColorRichness-btn" onclick="evaluateDimension('Color Richness')">Evaluate Color Richness</button>

<button id="generate-ColorRichness-suggestion-btn" onclick="generateSuggestion('Color Richness')">Generate Color Richness Suggestion</button>

<button id="submit-ColorRichness-score-Review-btn" onclick="submitScoreReview('Color Richness')">Submit Score & Review</button>

<button id="submit-ColorRichness-suggestion-btn" onclick="submitSuggestion('Color Richness')">Submit Suggestion</button>

<div id="loading-ColorRichness" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="ColorRichness-score" onchange="recordScoreChange('Color Richness')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="ColorRichness-Review" oninput="recordReviewChange('Color Richness')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="ColorRichness-suggestion" oninput="recordSuggestionChange('Color Richness')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="ColorContrast-section">

<h3>Color Contrast</h3>

<button id="evaluate-ColorContrast-btn" onclick="evaluateDimension('Color Contrast')">Evaluate Color Contrast</button>

<button id="generate-ColorContrast-suggestion-btn" onclick="generateSuggestion('Color Contrast')">Generate Color Contrast Suggestion</button>

<button id="submit-ColorContrast-score-Review-btn" onclick="submitScoreReview('Color Contrast')">Submit Score & Review</button>

<button id="submit-ColorContrast-suggestion-btn" onclick="submitSuggestion('Color Contrast')">Submit Suggestion</button>

<div id="loading-ColorContrast" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="ColorContrast-score" onchange="recordScoreChange('Color Contrast')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="ColorContrast-Review" oninput="recordReviewChange('Color Contrast')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="ColorContrast-suggestion" oninput="recordSuggestionChange('Color Contrast')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="LineCombination-section">

<h3>Line Combination</h3>

<button id="evaluate-LineCombination-btn" onclick="evaluateDimension('Line Combination')">Evaluate Line Combination</button>

<button id="generate-LineCombination-suggestion-btn" onclick="generateSuggestion('Line Combination')">Generate Line Combination Suggestion</button>

<button id="submit-LineCombination-score-Review-btn" onclick="submitScoreReview('Line Combination')">Submit Score & Review</button>

<button id="submit-LineCombination-suggestion-btn" onclick="submitSuggestion('Line Combination')">Submit Suggestion</button>

<div id="loading-LineCombination" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="LineCombination-score" onchange="recordScoreChange('Line Combination')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="LineCombination-Review" oninput="recordReviewChange('Line Combination')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="LineCombination-suggestion" oninput="recordSuggestionChange('Line Combination')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="LineTexture-section">

<h3>Line Texture</h3>

<button id="evaluate-LineTexture-btn" onclick="evaluateDimension('Line Texture')">Evaluate Line Texture</button>

<button id="generate-LineTexture-suggestion-btn" onclick="generateSuggestion('Line Texture')">Generate Line Texture Suggestion</button>

<button id="submit-LineTexture-score-Review-btn" onclick="submitScoreReview('Line Texture')">Submit Score & Review</button>

<button id="submit-LineTexture-suggestion-btn" onclick="submitSuggestion('Line Texture')">Submit Suggestion</button>

<div id="loading-LineTexture" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="LineTexture-score" onchange="recordScoreChange('Line Texture')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="LineTexture-Review" oninput="recordReviewChange('Line Texture')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="LineTexture-suggestion" oninput="recordSuggestionChange('Line Texture')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="PictureOrganization-section">

<h3>Picture Organization</h3>

<button id="evaluate-PictureOrganization-btn" onclick="evaluateDimension('Picture Organization')">Evaluate Picture Organization</button>

<button id="generate-PictureOrganization-suggestion-btn" onclick="generateSuggestion('Picture Organization')">Generate Picture Organization Suggestion</button>

<button id="submit-PictureOrganization-score-Review-btn" onclick="submitScoreReview('Picture Organization')">Submit Score & Review</button>

<button id="submit-PictureOrganization-suggestion-btn" onclick="submitSuggestion('Picture Organization')">Submit Suggestion</button>

<div id="loading-PictureOrganization" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="PictureOrganization-score" onchange="recordScoreChange('Picture Organization')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="PictureOrganization-Review" oninput="recordReviewChange('Picture Organization')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="PictureOrganization-suggestion" oninput="recordSuggestionChange('Picture Organization')" placeholder="Suggestion"></textarea>

</div>

<div class="dimension-section" id="Transformation-section">

<h3>Transformation</h3>

<button id="evaluate-Transformation-btn" onclick="evaluateDimension('Transformation')">Evaluate Transformation</button>

<button id="generate-Transformation-suggestion-btn" onclick="generateSuggestion('Transformation')">Generate Transformation Suggestion</button>

<button id="submit-Transformation-score-Review-btn" onclick="submitScoreReview('Transformation')">Submit Score & Review</button>

<button id="submit-Transformation-suggestion-btn" onclick="submitSuggestion('Transformation')">Submit Suggestion</button>

<div id="loading-Transformation" class="loading">Evaluating...</div>

<p>Score:</p>

<select id="Transformation-score" onchange="recordScoreChange('Transformation')">

<option value="not selected">not selected</option>

<option value="1">1</option>

<option value="2">2</option>

<option value="3">3</option>

<option value="4">4</option>

<option value="5">5</option>

</select>

<p>Review:</p>

<textarea id="Transformation-Review" oninput="recordReviewChange('Transformation')" placeholder="Review"></textarea>

<p>Suggestion:</p>

<textarea id="Transformation-suggestion" oninput="recordSuggestionChange('Transformation')" placeholder="Suggestion"></textarea>

</div>

</div>

</div>

<div class="right-column">

<div class="dimension-buttons">

<button onclick="scrollToDimension('Realistic-section')">Realistic</button>

<button onclick="scrollToDimension('Deformation-section')">Deformation</button>

<button onclick="scrollToDimension('Imagination-section')">Imagination</button>

<button onclick="scrollToDimension('ColorRichness-section')">Color Richness</button>

<button onclick="scrollToDimension('ColorContrast-section')">Color Contrast</button>

<button onclick="scrollToDimension('LineCombination-section')">Line Combination</button>

<button onclick="scrollToDimension('LineTexture-section')">Line Texture</button>

<button onclick="scrollToDimension('PictureOrganization-section')">Picture Organization</button>

<button onclick="scrollToDimension('Transformation-section')">Transformation</button>

</div>

</div>

</div>

<a href="{{ url\_for('upload\_file') }}">Upload another Artwork</a>

<script>

function scrollToDimension(dimensionId) {

const section = document.getElementById(dimensionId);

if (section) {

console.log(`Scrolling to section: ${dimensionId}`);

section.scrollIntoView({ behavior: 'smooth', block: 'start' });

} else {

console.log(`Section with id ${dimensionId} not found!`);

}

}

var initialReviews = {};

var initialScores = {};

var initialEntities = [];

var initialstyle = [] ;

var initialSuggestions = {};

document.addEventListener('DOMContentLoaded', () => {

document.querySelectorAll('#Entities #Entity').forEach(EntityElement => {

initialEntities.push(EntityElement.textContent.slice(0, -1).trim());

});

document.querySelectorAll('#Entities #styleEntity').forEach(EntityElement => {

initialstyle.push(EntityElement.textContent.slice(0, -1).trim());

});

document.querySelectorAll('select').forEach(selectElement => {

if (selectElement.value !== "not selected") {

selectElement.querySelector('option[value="not selected"]').disabled = true;

}

});

saveEntityActions();

});

function removeEntity(element, isStyleEntity = false) {

console.log("removeEntity");

console.log(isStyleEntity);

var EntityText = element.parentElement.textContent.slice(0, -1).trim();

element.parentElement.remove();

console.log("EntityText");

console.log(EntityText);

recordEntityChange('remove', EntityText, isStyleEntity);

}

function addEntity() {

var newEntityText = document.getElementById("newEntity").value.trim();

if (newEntityText !== "") {

var EntityContainer = document.getElementById("Entities");

var newEntity = document.createElement("div");

newEntity.className = "Entity";

newEntity.innerHTML = newEntityText + '<span class="remove" onclick="removeEntity(this)">x</span>';

EntityContainer.appendChild(newEntity);

document.getElementById("newEntity").value = "";

recordEntityChange('add', newEntityText);

}

}

function generateSuggestion(dimension) {

var imageData = "{{ image\_data }}";

var imageName = "{{ image\_name }}";

var normalizedDimension = dimension.replace(' ', '');

var suggestionElement = document.getElementById(`${normalizedDimension}-suggestion`);

var loadingElement = document.getElementById(`loading-${normalizedDimension}`);

var buttonElement = document.getElementById(`generate-${normalizedDimension}-suggestion-btn`);

if (buttonElement.disabled) {

return;

}

loadingElement.classList.add('show');

buttonElement.disabled = true;

fetch('/generate\_suggestion', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_data": imageData,

"dimension": dimension,

"image\_name": imageName

})

})

.then(response => response.json())

.then(data => {

suggestionElement.value = data.suggestion;

if (!userActions.suggestions[dimension]) {

userActions.suggestions[dimension] = {

original: data.suggestion,

current: data.suggestion

};

} else {

userActions.suggestions[dimension].original = data.suggestion;

userActions.suggestions[dimension].current = data.suggestion;

userActions.suggestions[dimension].added = '';

userActions.suggestions[dimension].removed = '';

}

loadingElement.classList.remove('show');

buttonElement.disabled = false;

})

.catch(error => {

console.error('Error:', error);

alert('An error occurred. Please try again.');

loadingElement.classList.remove('show');

buttonElement.disabled = false;

});

}

function evaluateDimension(dimension) {

var imageData = "{{ image\_data }}";

var imageName = "{{ image\_name }}";

var normalizedDimension = dimension.replace(' ', '');

var loadingElement = document.getElementById(`loading-${normalizedDimension}`);

var buttonElement = document.getElementById(`evaluate-${normalizedDimension}-btn`);

loadingElement.classList.add('show');

buttonElement.disabled = true;

fetch('/evaluate\_dimension', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_data": imageData,

"dimension": dimension,

"image\_name": imageName

})

})

.then(response => response.json())

.then(data => {

var scoreElement = document.getElementById(`${normalizedDimension}-score`);

var ReviewElement = document.getElementById(`${normalizedDimension}-Review`);

var suggestionElement = document.getElementById(`${normalizedDimension}-suggestion`);

scoreElement.value = data.score;

ReviewElement.value = data.Review;

suggestionElement.value = '';

if (!userActions.scores[dimension]) {

userActions.scores[dimension] = {

original: data.score,

current: data.score

};

} else {

userActions.scores[dimension].original = data.score;

userActions.scores[dimension].current = data.score;

}

if (!userActions.Reviews[dimension]) {

userActions.Reviews[dimension] = {

original: data.Review,

current: data.Review

};

} else {

userActions.Reviews[dimension].original = data.Review;

userActions.Reviews[dimension].current = data.Review;

}

scoreElement.querySelector('option[value="not selected"]').disabled = true;

loadingElement.classList.remove('show');

buttonElement.disabled = false;

updateRadarChart();

})

.catch(error => {

console.error('Error:', error);

alert('An error occurred. Please try again.');

loadingElement.classList.remove('show');

buttonElement.disabled = false;

});

}

var userActions = {

scores: {},

Reviews: {},

Entities: {

original: initialEntities,

added: [],

removed: []

},

style: {

original: initialstyle,

added: [],

removed: []

},

suggestions: {}

};

function recordScoreChange(dimension) {

var scoreElement = document.getElementById(`${dimension}-score`);

var currentScore = scoreElement.value;

if (currentScore !== "not selected") {

scoreElement.querySelector('option[value="not selected"]').disabled = true;

}

if (userActions.scores[dimension]) {

userActions.scores[dimension].current = currentScore;

} else {

userActions.scores[dimension] = {

initialGPTScore: currentScore,

original: currentScore,

current: currentScore

};

}

saveScoreReviewActions(dimension);

}

function recordReviewChange(dimension) {

var ReviewElement = document.getElementById(`${dimension}-Review`);

var currentReview = ReviewElement.value;

var originalReview = initialReviews[dimension] || '';

var removed = getRemovedText(originalReview, currentReview);

var added = getAddedText(originalReview, currentReview);

userActions.Reviews[dimension] = {

original: originalReview,

current: currentReview,

removed: removed,

added: added

};

saveScoreReviewActions(dimension);

}

function recordSuggestionChange(dimension) {

var suggestionElement = document.getElementById(`${dimension}-suggestion`);

var currentSuggestion = suggestionElement.value;

var originalSuggestion = initialSuggestions[dimension] || '';

var removed = getRemovedText(originalSuggestion, currentSuggestion);

var added = getAddedText(originalSuggestion, currentSuggestion);

if (!userActions.suggestions) {

userActions.suggestions = {};

}

userActions.suggestions[dimension] = {

original: originalSuggestion,

current: currentSuggestion,

removed: removed,

added: added

};

saveSuggestionActions(dimension);

}

function saveScoreReviewActions(dimension) {

var imageName = "{{ image\_name }}";

fetch('/save\_user\_actions', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_name": imageName,

"user\_actions": {

"scores": { [dimension]: userActions.scores[dimension] },

"Reviews": { [dimension]: userActions.Reviews[dimension] }

}

})

})

.then(response => response.json())

.then(data => {

console.log('Score and Review actions saved:', data);

})

.catch(error => console.error('Error saving score and Review actions:', error));

}

function saveSuggestionActions(dimension) {

var imageName = "{{ image\_name }}";

fetch('/save\_user\_actions', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_name": imageName,

"user\_actions": {

"suggestions": { [dimension]: userActions.suggestions[dimension] }

}

})

})

.then(response => response.json())

.then(data => {

console.log('Suggestion actions saved:', data);

})

.catch(error => console.error('Error saving suggestion actions:', error));

}

function getRemovedText(original, current) {

return original.split('').filter(char => !current.includes(char)).join('');

}

function getAddedText(original, current) {

return current.split('').filter(char => !original.includes(char)).join('');

}

function recordEntityChange(action, Entity, isStyleEntity = false) {

console.log("isStyleEntity");

console.log(isStyleEntity);

if (isStyleEntity) {

if (!userActions.style) {

userActions.style = { original: [], added: [], removed: [] };

}

if (action === 'add') {

userActions.style.added.push(Entity);

} else if (action === 'remove') {

userActions.style = { original: [Entity], added: [], removed: [Entity] };

}

} else {

if (action === 'add') {

userActions.Entities.added.push(Entity);

} else if (action === 'remove') {

userActions.Entities.removed.push(Entity);

}

}

saveEntityActions();

}

function saveEntityActions() {

var imageName = "{{ image\_name }}";

fetch('/save\_user\_actions', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_name": imageName,

"user\_actions": {

"Entities": userActions.Entities,

"style": userActions.style

}

})

})

.then(response => response.json())

.then(data => {

console.log('Entity and style actions saved:', data);

})

.catch(error => console.error('Error saving Entity and style actions:', error));

}

function updateRadarChart() {

fetch('/update\_radar\_chart', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

}

})

.then(response => response.json())

.then(data => {

document.getElementById('radar-chart').src = `data:image/png;base64,${data.radar\_chart}`;

})

.catch(error => console.error('Error:', error));

}

function submitScoreReview(dimension) {

var imageName = "{{ image\_name }}";

console.log(`Submitting score\_Review for dimension: ${dimension}`);

fetch('/submit\_score\_Review', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_name": imageName,

"dimension": dimension

})

})

.then(response => response.json())

.then(data => {

console.log(`Score and Review for ${dimension} submitted: `, data);

alert(`Score and Review for ${dimension} submitted successfully.`);

})

.catch(error => {

console.error('Error submitting score and Review:', error);

});

}

function submitSuggestion(dimension) {

var imageName = "{{ image\_name }}";

console.log(`Submitting suggestion for dimension: ${dimension}`);

fetch('/submit\_suggestion', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

"image\_name": imageName,

"dimension": dimension

})

})

.then(response => response.json())

.then(data => {

console.log(`Suggestion for ${dimension} submitted: `, data);

alert(`Suggestion for ${dimension} submitted successfully.`);

})

.catch(error => {

console.error('Error submitting suggestion:', error);

});

}

</script>

</body>

</html>

import os

import json

import numpy as np

import pandas as pd

from scipy.stats import spearmanr

# Define dimensions

dimensions = [

"Realistic", "Deformation", "Imagination", "Color Richness",

"Color Contrast", "Line Combination", "Line Texture",

"Picture Organization", "Transformation"

]

# Load JSON file

def load\_json\_data(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return json.load(f)

except FileNotFoundError:

print(f"File not found: {file\_path}")

return None

# Extract all `original` and `current` score sequences for each dimension

def extract\_scores(score\_comment\_dir, output\_file\_scores):

score\_results = []

for image\_num in range(1, 21): # Adjust based on actual range

print(f"Processing image number: {image\_num}")

for dimension in dimensions:

# Process score\_Review file

score\_comment\_file = os.path.join(score\_comment\_dir, f"{image\_num}.jpg\_{dimension}\_score\_Review.json")

score\_comment\_data = load\_json\_data(score\_comment\_file)

if score\_comment\_data:

print(f"Processing file: {score\_comment\_file}")

# Extract `original` and `current` scores for all rounds

for round\_data in score\_comment\_data:

if round\_data["round"] == 1:

continue # Skip initial round 1 data

gpt\_score = round\_data['data']['scores']['original']

user\_score = round\_data['data']['scores']['current']

if gpt\_score is not None and user\_score is not None:

score\_results.append({

'image': f"{image\_num}.jpg",

'dimension': dimension,

'original': float(gpt\_score),

'current': float(user\_score)

})

# Save the scores to a DataFrame and output to an Excel file

scores\_df = pd.DataFrame(score\_results)

scores\_df.to\_excel(output\_file\_scores, index=False)

print(f"Original and Current scores have been saved to: {output\_file\_scores}")

# Calculate SC (Spearman) for score consistency

def get\_sc(original\_scores, current\_scores):

spearman\_corr, \_ = spearmanr(original\_scores, current\_scores)

return spearman\_corr

# Calculate SD (Score Difference)

def get\_sd(original\_scores, current\_scores):

score\_diff = np.abs(original\_scores - current\_scores)

avg\_sd = score\_diff.mean()

return avg\_sd

# Process correlations and score differences for each dimension

def calculate\_sc\_sd(output\_file\_scores):

# Load the saved scores

scores\_df = pd.read\_excel(output\_file\_scores)

# Group by dimension and calculate the SC and SD for each

sc\_sd\_results = []

for dimension in dimensions:

dim\_scores = scores\_df[scores\_df['dimension'] == dimension]

original\_scores = dim\_scores['original']

current\_scores = dim\_scores['current']

# Calculate SC (Spearman correlation) and SD (Average Difference)

sc\_value = get\_sc(original\_scores, current\_scores)

sd\_value = get\_sd(original\_scores, current\_scores)

sc\_sd\_results.append({

'dimension': dimension,

'SC': sc\_value,

'SD': sd\_value

})

# Save the SC and SD results

sc\_sd\_df = pd.DataFrame(sc\_sd\_results)

sc\_sd\_df.to\_excel("SC\_SD\_Result.xlsx", index=False)

print(f"SC and SD results have been saved to: SC\_SD\_Results.xlsx")

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

score\_comment\_directory = "userActionsEveryRounds/score\_Review" # Path to JSON files

output\_file\_scores = "SC\_SD\_Results.xlsx"

# Calculate SC (Spearman) and SD (Average Difference)

calculate\_sc\_sd(output\_file\_scores)

import os

import json

import pandas as pd

folder\_path = "userActions/Entities"

def get\_ass(folder\_path, output\_file):

results = [] # Store file names and whether the deletion was detected (1 or 0)

N = 0 # N: Total number of files

D = 0 # D: Number of deletion operations

# Loop through all files in the folder

for file\_name in os.listdir(folder\_path):

if file\_name.endswith('\_labels.json'):

file\_path = os.path.join(folder\_path, file\_name)

# Load the JSON file

with open(file\_path, 'r', encoding='utf-8') as f:

data = json.load(f)

# Check the 'removed' field in the 'style'

style\_removed = data.get("style", {}).get("removed", [])

if style\_removed:

results.append([file\_name, 0]) # 0 means incorrect recognition (deletion detected)

D += 1 # If the 'removed' list has content, count as one deletion

else:

results.append([file\_name, 1]) # 1 means correct recognition (no deletion detected)

N += 1 # Count the number of files

if N == 0:

print("No matching files found.")

return

# Calculate ASS (Artistic Style Sensitivity)

ass = 1 - (D / N)

print(f"Artistic Style Sensitivity (ASS) index is: {ass:.2f}")

# Save the results into a DataFrame

df = pd.DataFrame(results, columns=["File Name", "Correct Recognition (1=Correct, 0=Incorrect)"])

# Export the results to an Excel file

df.to\_excel(output\_file, index=False)

print(f"Results have been saved to: {output\_file}")

# Main function call

if \_\_name\_\_ == "\_\_main\_\_":

output\_file = "ASS\_Result.xlsx"

get\_ass(folder\_path, output\_file)

import os

import json

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

# Define the normalize function

def normalize(value, min\_value, max\_value):

if max\_value - min\_value == 0:

return 0

return (value - min\_value) / (max\_value - min\_value)

# Define the 9 dimension names

dimensions = [

"Realistic", "Deformation", "Imagination", "Color Richness",

"Color Contrast", "Line Combination", "Line Texture",

"Picture Organization", "Transformation"

]

# Load JSON file

def load\_json\_data(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return json.load(f)

except FileNotFoundError:

print(f"File not found: {file\_path}")

return None

# Calculate TAR (Text Acceptance Rate)

def get\_tar(text\_data):

total\_tar = 0

valid\_rounds = 0 # Record valid rounds

for round\_data in text\_data:

if not isinstance(round\_data, dict):

continue # Ensure round\_data is a dictionary

if round\_data.get("round") == 1:

print("Skipping round 1 data")

continue # Skip round 1

# Process the 'reviews' field

reviews\_data = round\_data.get('data', {}).get('Reviews', None)

if reviews\_data:

original = reviews\_data.get('original', "")

added = reviews\_data.get('added', "")

removed = reviews\_data.get('removed', "")

if original:

len\_original = len(original)

len\_added = len(added)

len\_removed = len(removed)

print("Original", len\_original,"added", len\_added,"removed", len\_removed)

denominator = len\_added + len\_original

if denominator > 0:

tar\_round = (len\_original - len\_removed) / denominator

total\_tar += tar\_round

valid\_rounds += 1

# Process the 'suggestions' field

suggestions\_data = round\_data.get('data', {}).get('suggestions', None)

if suggestions\_data:

original = suggestions\_data.get('original', "")

added = suggestions\_data.get('added', "")

removed = suggestions\_data.get('removed', "")

if original:

len\_original = len(original)

len\_added = len(added)

len\_removed = len(removed)

denominator = len\_added + len\_original

if denominator > 0:

tar\_round = (len\_original - len\_removed) / denominator

total\_tar += tar\_round

valid\_rounds += 1

if valid\_rounds == 0:

print("No valid rounds")

return np.nan # Return NaN if no valid rounds

# Calculate the average TAR

average\_tar = total\_tar / valid\_rounds

print("total\_tar", total\_tar,"valid\_rounds", valid\_rounds)

print("average\_tar",average\_tar)

return average\_tar

# Calculate TS (Text Similarity)

def get\_ts(text\_data):

gpt\_texts = []

user\_texts = []

for round\_data in text\_data:

if round\_data["round"] == 1:

continue

# Process the 'reviews' field

reviews\_data = round\_data['data'].get('Reviews', {})

if reviews\_data:

gpt\_text = reviews\_data.get('original', "")

user\_text = reviews\_data.get('current', "")

# if gpt\_text.strip() and user\_text.strip():

gpt\_texts.append(gpt\_text)

user\_texts.append(user\_text)

# Process the 'suggestions' field

suggestions\_data = round\_data['data'].get('suggestions', {})

if suggestions\_data:

gpt\_text = suggestions\_data.get('original', "")

user\_text = suggestions\_data.get('current', "")

if gpt\_text.strip() and user\_text.strip():

gpt\_texts.append(gpt\_text)

user\_texts.append(user\_text)

if not gpt\_texts or not user\_texts:

return np.nan # Return NaN if no data

# Use bag-of-words model to analyze based on words, not characters

vectorizer = CountVectorizer(analyzer='word', token\_pattern=r"(?u)\b\w+\b").fit([gpt\_texts[-1], user\_texts[-1]])

# Vectorize the GPT and user texts

gpt\_vector = vectorizer.transform([gpt\_texts[-1]]).toarray()

user\_vector = vectorizer.transform([user\_texts[-1]]).toarray()

if gpt\_vector.shape[1] > 1 and user\_vector.shape[1] > 1:

similarity = cosine\_similarity(gpt\_vector, user\_vector)[0][0]

return similarity # Return the unnormalized cosine similarity

return np.nan # Return NaN if no data

# Process directory and calculate TAR and TS

def process\_directory(score\_comment\_dir, suggestion\_dir, output\_file\_tar\_rev, output\_file\_tar\_sug, output\_file\_ts\_rev,output\_file\_ts\_sug):

tar\_results\_review = []

ts\_results\_review = []

tar\_results\_suggestion = []

ts\_results\_suggestion = []

for image\_num in range(1, 21): # Adjust based on actual range

tar\_values\_review = []

ts\_values\_review = []

tar\_values\_suggestion = []

ts\_values\_suggestion = []

for dimension in dimensions:

# Process score\_Review file

score\_comment\_file = os.path.join(score\_comment\_dir, f"{image\_num}.jpg\_{dimension}\_score\_Review.json")

score\_comment\_data = load\_json\_data(score\_comment\_file)

print("######TAR TS Review########")

print(score\_comment\_file)

if score\_comment\_data:

# Calculate TAR and TS for reviews

tar\_value\_review = get\_tar(score\_comment\_data)

print("tar\_value\_review",tar\_value\_review)

ts\_value\_review = get\_ts(score\_comment\_data)

print("ts\_value\_review",ts\_value\_review)

# Store the review results

tar\_values\_review.append(tar\_value\_review)

ts\_values\_review.append(ts\_value\_review)

# Process suggestions file

suggestion\_file = os.path.join(suggestion\_dir, f"{image\_num}.jpg\_{dimension}\_suggestion.json")

print(suggestion\_file)

suggestion\_data = load\_json\_data(suggestion\_file)

print("######TAR TS Suggestion########")

if suggestion\_data:

# Calculate TAR and TS for suggestions

tar\_value\_suggestion = get\_tar(suggestion\_data)

ts\_value\_suggestion = get\_ts(suggestion\_data)

print("tar\_value\_Suggestion", tar\_value\_review)

print("tar\_value\_Suggestion", tar\_value\_review)

# Store the suggestion results

tar\_values\_suggestion.append(tar\_value\_suggestion)

ts\_values\_suggestion.append(ts\_value\_suggestion)

# Append results for TAR and TS

tar\_results\_review.append([f"{image\_num}.jpg"] + tar\_values\_review)

ts\_results\_review.append([f"{image\_num}.jpg"] + ts\_values\_review)

tar\_results\_suggestion.append([f"{image\_num}.jpg"] + tar\_values\_suggestion)

ts\_results\_suggestion.append([f"{image\_num}.jpg"] + ts\_values\_suggestion)

# Define columns for Review and Suggestion TAR/TS for each dimension

Review\_tar\_columns = ["File Name"] + [f"{dim}\_Review\_TAR" for dim in dimensions]

Review\_ts\_columns = ["File Name"] + [f"{dim}\_Review\_TS" for dim in dimensions]

Suggestion\_tar\_columns = ["File Name"] + [f"{dim}\_Suggestion\_TAR" for dim in dimensions]

Suggestion\_ts\_columns = ["File Name"] + [f"{dim}\_Suggestion\_TS" for dim in dimensions]

# Save TAR results to Excel

tar\_df\_review = pd.DataFrame(tar\_results\_review, columns=Review\_tar\_columns)

tar\_df\_review.to\_excel(output\_file\_tar\_rev, index=False)

print(f"TAR results have been saved to: {output\_file\_tar\_rev}")

tar\_df\_suggestion = pd.DataFrame(tar\_results\_suggestion, columns=Suggestion\_tar\_columns)

tar\_df\_suggestion.to\_excel(output\_file\_tar\_sug, index=False)

print(f"TAR results have been saved to: {output\_file\_tar\_sug}")

# Save TS results to Excel

ts\_df\_review = pd.DataFrame(ts\_results\_review, columns=Review\_ts\_columns)

ts\_df\_review.to\_excel(output\_file\_ts\_rev, index=False)

print(f"TS results have been saved to: {output\_file\_ts\_rev}")

ts\_df\_sug = pd.DataFrame(ts\_results\_suggestion, columns=Suggestion\_ts\_columns)

ts\_df\_sug.to\_excel(output\_file\_ts\_sug, index=False)

print(f"TS results have been saved to: {output\_file\_ts\_sug}")

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

score\_comment\_directory = "userActionsEveryRounds/score\_Review" # Path to score review JSON files

suggestion\_directory = "userActionsEveryRounds/suggestion" # Path to suggestion JSON files

output\_file\_tar\_rev = "TAR\_Results\_rev.xlsx"

output\_file\_tar\_sug = "TAR\_Results\_sug.xlsx"

output\_file\_ts\_rev = "TS\_Results\_rev.xlsx"

output\_file\_ts\_sug = "TS\_Results\_sug.xlsx"

process\_directory(score\_comment\_directory, suggestion\_directory,

output\_file\_tar\_rev, output\_file\_tar\_sug,

output\_file\_ts\_rev,output\_file\_ts\_sug)

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<title>ArtMentor</title>

<style>

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: radial-gradient(ellipse at center,#fffeea 0%,#fffeea 35%,#b7e8eb 100%);

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.container {

background-color: #fffbf2;

padding: 25px 50px;

border-radius: 15px;

box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);

text-align: center;

position: relative;

border: 2px solid #d4b483;

max-width: 400px;

width: 100%;

}

.brand {

font-family: 'Brush Script MT', cursive;

font-size: 2em;

color: #8a6d3b;

position: absolute;

top: -35px;

left: 50%;

transform: translateX(-50%);

background-color: #fffbf2;

padding: 0 10px;

}

h1 {

margin-bottom: 25px;

color: #8a6d3b;

font-size: 1.8em;

}

form {

display: flex;

flex-direction: column;

align-items: center;

}

.file-input {

margin-bottom: 25px;

position: relative;

width: 100%;

}

.file-input input[type="file"] {

opacity: 0;

position: absolute;

width: 100%;

height: 100%;

top: 0;

left: 0;

cursor: pointer;

}

.file-input-label {

display: block;

padding: 12px;

background-color: #f3e5c8;

color: #704214;

border-radius: 8px;

cursor: pointer;

text-align: center;

transition: background-color 0.3s ease;

border: 1px solid #c7a17a;

}

.file-input-label:hover {

background-color: #e8c496;

}

input[type="submit"] {

padding: 12px 25px;

background-color: #a4c49a;

color: #3e4a35;

border: none;

border-radius: 8px;

cursor: pointer;

transition: background-color 0.3s ease;

font-size: 1em;

}

input[type="submit"]:hover {

background-color: #8fa87e;

}

.loading-overlay {

display: none;

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

background-color: rgba(255, 255, 255, 0.85);

border-radius: 15px;

justify-content: center;

align-items: center;

}

.loading-overlay.active {

display: flex;

}

.loading-spinner {

border: 6px solid #f3f3f3;

border-radius: 50%;

border-top: 6px solid #8a6d3b;

width: 60px;

height: 60px;

animation: spin 1s linear infinite;

}

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

</style>

</head>

<body>

<div class="container">

<div class="brand">ArtMentor</div>

<h1>Upload an Artwork for Analysis</h1>

<form id="uploadForm" method="post" enctype="multipart/form-data">

<div class="file-input">

<label class="file-input-label" for="file">Choose an Artwork</label>

<input type="file" id="file" name="file" required>

</div>

<input type="submit" value="Upload">

</form>

<div class="loading-overlay" id="loadingOverlay">

<div class="loading-spinner"></div>

</div>

</div>

<script>

const form = document.getElementById('uploadForm');

const loadingOverlay = document.getElementById('loadingOverlay');

form.addEventListener('submit', function() {

loadingOverlay.classList.add('active');

});

</script>

</body>

</html>