Cheat Sheet: Integrating Visual and Video Modalities

Package/Method	Description	Code Example
Base64 response format	Instead of returning URLs, you can get images as base64 data for immediate use without downloading from a URL. Useful when you need to process or store the images directly.	`import base64` `from PIL import Image` `import io` `response = client.images.generate(` `  model="dall-e-2",` `  prompt="a white siamese cat",` `  size="512x512",` `  response_format="b64_json",  # Get base64 instead of URL` `  n=1,` `)` `// Convert base64 to image` `image_data = base64.b64decode(response.data[0].b64_json)` `image = Image.open(io.BytesIO(image_data))` `image.show()  # Display the image`
Credentials setup	Sets up the credentials for accessing the watsonx API. The api_key is not needed in the lab environment, and the project_id is preset.	`from ibm_watsonx_ai import Credentials` `import os` `credentials = Credentials(` `  url="https://us-south.ml.cloud.ibm.com",` `  )` `project_id="skills-network"`
DALL-E 2 image generation	Uses DALL-E 2 to generate an image based on a text prompt. DALL-E 2 supports generations, edits, and variations, simultaneously allowing up to 10 images.	`response = client.images.generate(` `  model="dall-e-2",` `  prompt="a white siamese cat",` `  size="1024x1024",` `  quality="standard",` `  n=1,` `)` `url = response.data[0].url` `display.Image(url=url, width=512)`
DALL-E 3 image generation	Uses DALL-E 3 to generate higher quality images. DALL-E 3 only supports image generation (no edits or variations) but produces more detailed, accurate images.	`response = client.images.generate(` `  model="dall-e-3",` `  prompt="a white siamese cat",` `  size="1024x1024",` `  quality="standard",` `  n=1,` `)` `url = response.data[0].url` `display.Image(url=url, width=512)`

Effective prompting	Tips for crafting effective prompts to get better results from DALL-E models: • Be specific and detailed in your descriptions • Include artistic style references • Specify lighting, perspective, and composition • Add context or setting information	`// Basic prompt` `prompt = "a cat"` `// Improved detailed prompt` `prompt = “a fluffy white siamese cat with blue eyes sitting on a window sill, golden hour lighting, soft shadows, shallow depth of field, professional photography style”` `// Artistic style prompt` `prompt = “a white siamese cat in the style of a Renaissance oil painting, dramatic lighting, rich colors, detailed fur texture”`
File download	Function to download an image file from a URL if it doesn't already exist locally.	`import requests` `def load_file(filename, url):` `  # Download file if it doesn’t already exist` `  if not os.path.isfile(filename):` `  print(“Downloading file”)` `  response = requests.get(url, stream=True)` `  if response.status_code == 200:` `  with open(filename, ‘wb’) as f:` `  f.write(response.content)` `  else:` `  print(“Failed to download file. Status code:”, response.status_code)` `  else:` `  print(“File already exists”)`
Image captioning	Loop through the images to see the text descriptions produced by the model in response to the query, "Describe the photo".	`user_query = "Describe the photo"` `for i in range(len(encoded_images)):` `  image = encoded_images[i]` `  response = generate_model_response(image, user_query)` `  // Print the response with a formatted description` `  print(f"Description for image {i + 1}: {response}/n/n")`
Image display	Displays an image in the notebook using IPython's display functionality.	`from IPython.display import Image` `Image(filename=filename_tim, width=300)`
Image encoding	Encodes an image to base64 format for inclusion in the model request. This is necessary because JSON is text-based and doesn't support	`import base64` `import requests` `def encode_images_to_base64(image_urls):` `  encoded_images = []` `  for url in image_urls:` `  response = requests.get(url)` `  if response.status_code == 200:` `  encoded_image = base64.b64encode(response.content).decode(“utf-8”)` `  encoded_images.append(encoded_image)` `  print(type(encoded_image))`

	binary data directly.	else: print(f"Warning: Failed to fetch image from {url} (Status code: {response.status_code})") encoded_images.append(None) return encoded_images
Message formatting	Creates a structured message containing both text and image data to send to the model.	messages = [{ "role": "user", "content": [{ "type": "text", "text": question }, { "type": "image_url", "image_url": { "url": "" + encoded_string, } }] }] return messages
Model invocation	Sends the formatted message to the model and receives a response with an analysis of the image.	response = model.chat(messages=my_message_1) print(response["choices"][0]["message"]["content"])
Model initialization	Initializes the vision model with specific parameters for text generation.	from ibm_watsonx_ai.foundation_models.schema import TextChatParameters from ibm_watsonx_ai.foundation_models import ModelInference model_id = 'ibm/granite-vision-3-2-2b' params = TextChatParameters(temperature=0.2, top_p=0.5,) model = ModelInference(model_id=model_id, credentials=credentials, project_id=project_id, params=params)
Multiple images (DALL-E 2)	Generate multiple images at once with DALL-E 2 using the 'n' parameter. DALL-E 2 can generate up to 10 images in a single request.	response = client.images.generate(model="dall-e-2", prompt="a white siamese cat", size="1024x1024", quality="standard", n=4, # Generate 4 different images) // Access all generated images for i, image_data in enumerate(response.data): print(f"URL for image {i+1}: {image_data.url}") display.Image(url=image_data.url, width=256)

OpenAI client initialization	Creates an instance of the OpenAI client to interact with the API.	`from openai import OpenAI` `from IPython import display` `client = OpenAI()`
Object dectection	Ask the model to define objects from a specific image.	`image = encoded_images[1]` `user_query = "How many cars are in this image?"` `print("User Query: ", user_query)` `print("Model Response: ", generate_model_response(image, user_query))`
pip install	Installs the necessary Python libraries required for working with watsonx and vision models.	`%pip install ibm-watsonx-ai==1.1.20 image==1.5.33 requests==2.32.0`
Quality options	Quality settings for generated images: • DALL-E 2: Only supports "standard" • DALL-E 3: Supports "standard" (default) and "hd" for enhanced detail	`// DALL-E 3 with high-definition quality` `response = client.images.generate(` `  model="dall-e-3",` `  prompt="a mountain landscape",` `  size="1024x1024",` `  quality="hd",` `  n=1,` `)`
Saving generated images	Save the generated images to your local filesystem for later use.	`import requests` `// Save from URL` `response = client.images.generate(` `  model="dall-e-2",` `  prompt="a white siamese cat",` `  size="1024x1024",` `)` `url = response.data[0].url` `image_data = requests.get(url).content` `with open("generated_cat.jpg", "wb") as f:` `  f.write(image_data)` `print("Image saved to generated_cat.jpg")`

Size options	Different size options available for DALL-E models: • DALL-E 2: 256x256, 512x512, 1024x1024 • DALL-E 3: 1024x1024, 1024x1792, 1792x1024	```// DALL-E 2 with smaller size response = client.images.generate(model="dall-e-2", prompt="a white siamese cat", size="512x512", quality="standard", n=1,) // DALL-E 3 with widescreen format response = client.images.generate(model="dall-e-3", prompt="a beautiful landscape", size="1792x1024", quality="standard", n=1,)```

Author

Hailey Quach