

Cheat Sheet: Advanced Multimodal Applications

| Package/Method                     | Description   | Code Example   |
|------------------------------------|---|--|
| Basic image querying               | Create a simple function to send an image to a vision model and get a response to a general question about the image. | <pre>def generate_model_response(encoded_image, user_query,                              assistant_prompt="You are a helpful assistant. Answer the following user query in 1 or 2 sent                              """Send image and query to model and get response."""     messages = [         {             "role": "user",             "content": [                 {                     "type": "text",                     "text": assistant_prompt + user_query                 },                 {                     "type": "image_url",                     "image_url": {                         "url": "data:image/jpeg;base64," + encoded_image,                     }                 }             ]         }     ]     response = model.chat(messages=messages)     return response['choices'][0]['message']['content']  // Example usage: user_query = "Describe the photo" response = generate_model_response(encoded_images[0], user_query) print("Description:", response)</pre> |
| Basic object detection             | Use the vision model to detect and count objects in images by asking specific questions.                              | <pre>// Detection examples for various use cases image = encoded_images[1] // Select second image  // Count objects result = generate_model_response(     image,     "How many cars are in this image?" ) print("Cars detected:", result)  // Examine details result = generate_model_response(     image,     "What color is the woman's jacket in this image?" ) print("Clothing analysis:", result)  // Read text from images result = generate_model_response(     encoded_images[3], # Nutrition label image     "How much sodium is in this product?" ) print("Sodium content:", result)</pre>   |
| Creating messages for vision model | Format a request with both text and image data to send to the multimodal model.                                       | <pre>def create_vision_message(prompt, encoded_image):     messages = [         {             "role": "user",             "content": [                 {                     "type": "text",                     "text": prompt                 },                 {                     "type": "image_url",                     "image_url": {                         "url": "data:image/jpeg;base64," + encoded_image,                     }                 }             ]         }     ]     return messages</pre>   |

|   |   |  |
|---|---|--|
|   |   |  |
| Environment setup                       | Create and activate a virtual environment, then install necessary packages for multimodal applications. | <pre>python3.11 -m venv venv source venv/bin/activate pip install ibm-watsonx-ai==1.1.20 image==1.5.33 flask requests==2.32.0 pip install torch torchvision scikit-learn pillow gradio</pre>   |
| Fashion analysis prompting              | Specialized prompting for fashion analysis with structured output for retail applications.              | <pre>def generate_fashion_response(user_image_base64, matched_row, all_items,                              similarity_score, threshold=0.8):     """Generate fashion-specific analysis with product details."""     // Generate list of items with prices and links     items_list = []     for _, row in all_items.iterrows():         item_str = f"{row['Item Name']} ({row['Price']}): {row['Link']}"         items_list.append(item_str)     // Join with proper formatting     items_description = "\n".join([f"- {item}" for item in items_list])     if similarity_score &gt;= threshold:         // Prompt for exact matches         assistant_prompt = f"""         You're conducting a professional retail catalog analysis.         Focus exclusively on professional fashion analysis.         ITEM DETAILS (always include this section):         {items_description}         Please:         1. Identify and describe clothing items objectively (colors, patterns, materials)         2. Categorize the overall style (business, casual, etc.)         3. Include the ITEM DETAILS section at the end         Use formal, clinical language for a professional catalog.         """     else:         // Prompt for similar but not exact matches         assistant_prompt = f"""         You're conducting a professional retail catalog analysis.         Focus exclusively on professional fashion analysis.         SIMILAR ITEMS (always include this section):         {items_description}         Please:         1. Note these are similar but not exact items         2. Identify clothing elements objectively         3. Include the SIMILAR ITEMS section at the end         Use formal, clinical language for a professional catalog.         """     // Generate and return response     return generate_model_response(user_image_base64,                                    "Analyze this outfit",                                    assistant_prompt)</pre> |
| Flask integration for vision AI web app | Basic Flask setup to create a web application with vision AI capabilities.                              | <pre>from flask import Flask, render_template, request app = Flask(__name__) @app.route("/", methods=["GET", "POST"]) def index():     if request.method == "POST":         # Retrieve user inputs         user_query = request.form.get("user_query")         uploaded_file = request.files.get("file")         if uploaded_file:             # Process the uploaded image             encoded_image = input_image_setup(uploaded_file)             # Generate the model's response             response = generate_model_response(encoded_image, user_query, assistant_prompt)             # Render the result             return render_template("index.html", user_query=user_query, response=response)     return render_template("index.html")  if __name__ == "__main__":     app.run(debug=True)</pre>   |
| Image encoding from URLs                | Load and encode multiple images from URLs to  | <pre>import requests import base64  // Define image URLs url_image_1 = 'https://example.com/image1.jpg' url_image_2 = 'https://example.com/image2.jpg' image_urls = [url_image_1, url_image_2]</pre>   |

|                             |  |   |
|-----------------------------|--|---|
|                             | base64 format for batch processing with vision models.   | <pre>// Encode all images encoded_images = [] for url in image_urls:     encoded_images.append(         base64.b64encode(             requests.get(url).content         ).decode( "utf-8" )     )</pre>   |
| Image encoding from uploads | Convert an uploaded image file to base64 format for inclusion in a request to a vision model.              | <pre>import base64 from PIL import Image from io import BytesIO  def input_image_setup(uploaded_file):     if uploaded_file is not None:         // Read file into bytes         bytes_data = uploaded_file.read()         // Encode image to base64 string         encoded_image = base64.b64encode(bytes_data).decode( "utf-8" )         return encoded_image     else:         raise FileNotFoundError( "No file uploaded" )</pre>   |
| Nutrition analysis prompt   | Detailed prompt template for analyzing food images with structured output focusing on nutritional content. | <pre>def generate_nutrition_response(encoded_image, user_query):     """Generate detailed nutrition analysis response."""     assistant_prompt = """     You are an expert nutritionist. Your task is to analyze the     food items displayed in the image and provide a detailed     nutritional assessment using the following format:     1. **Identification**: List each identified food item clearly,        one per line.     2. **Portion Size &amp; Calorie Estimation**: For each identified        food item, specify the portion size and provide an        estimated number of calories. Use bullet points with        the following structure:        * **[Food Item]**: [Portion Size], [Number of Calories] calories        Example:        * **Salmon**: 6 ounces, 210 calories        * **Asparagus**: 3 spears, 25 calories     3. **Total Calories**: Provide the total number of calories        for all food items.        Example:        Total Calories: 235 calories     4. **Nutrient Breakdown**: Include a breakdown of key nutrients        such as **Protein**, **Carbohydrates**, **Fats**, **Vitamins**,        and **Minerals**. Use bullet points for each nutrient.        Example:        * **Protein**: Salmon (35g), Asparagus (3g) = 38g total        * **Carbohydrates**: Asparagus (5g) = 5g total     5. **Health Evaluation**: Evaluate the healthiness of the        meal in one paragraph.     6. **Disclaimer**: Include the following exact text:        The nutritional information and calorie estimates provided        are approximate and are based on general food data.        Actual values may vary depending on factors such as        portion size, specific ingredients, preparation methods,        and individual variations. For precise dietary advice or        medical guidance, consult a qualified nutritionist or        healthcare provider.     """     return generate_model_response(encoded_image, user_query, assistant_prompt)</pre> |

|                              |   |  |
|------------------------------|---|--|
| Similarity matching          | Find the closest matching image in a dataset based on cosine similarity of vector embeddings.   | <pre> from sklearn.metrics.pairwise import cosine_similarity  def find_closest_match(user_vector, dataset):     """ Find closest match based on cosine similarity. """     try:         // Stack all vectors from dataset         dataset_vectors = np.vstack(dataset[ 'Embedding' ].dropna().values)         // Calculate similarities         similarities = cosine_similarity(user_vector.reshape(1, -1), dataset_vectors)         // Find highest similarity index         closest_index = np.argmax(similarities)         similarity_score = similarities[0][closest_index]         // Get corresponding dataset row         closest_row = dataset.iloc[closest_index]         return closest_row, similarity_score     except Exception as e:         print(f" Error finding closest match: {e} ")         return None, None </pre>  |
| Vector embeddings for images | Convert images to vector embeddings for similarity matching using a pre-trained ResNet50 model. | <pre> import torch import torchvision.transforms as transforms from torchvision.models import resnet50 import numpy as np  class ImageProcessor:     def __init__(self, image_size=(224, 224),                   norm_mean=[0.485, 0.456, 0.406],                   norm_std=[0.229, 0.224, 0.225]):         self.device = torch.device( "cuda" if torch.cuda.is_available() else "cpu" )         self.model = resnet50(pretrained=True).to(self.device)         self.model.eval() // Set model to evaluation mode         // Image preprocessing pipeline         self.preprocess = transforms.Compose([             transforms.Resize(image_size),             transforms.ToTensor(),             transforms.Normalize(mean=norm_mean, std=norm_std),         ])     def encode_image(self, image_input, is_url=True):         try:             if is_url:                 // Fetch image from URL                 response = requests.get(image_input)                 image = Image.open(BytesIO(response.content)).convert( "RGB" )             else:                 // Load from local file                 image = Image.open(image_input).convert( "RGB" )             // Convert image to Base64             buffered = BytesIO()             image.save(buffered, format=" JPEG" )             base64_string = base64.b64encode(buffered.getvalue()).decode( "utf-8" )             // Get feature vector using ResNet50             input_tensor = self.preprocess(image).unsqueeze(0).to(self.device)             with torch.no_grad():                 features = self.model(input_tensor)             // Convert to NumPy array             feature_vector = features.cpu().numpy().flatten()             return { "base64" : base64_string, "vector" : feature_vector }         except Exception as e:             print(f" Error encoding image: {e} ")             return { "base64" : None, "vector" : None } </pre> |
| Vision model initialization  | Set up credentials and initialize the Llama 3.2 Vision Instruct model through watsonx.ai.       | <pre> from ibm_watsonx_ai import Credentials from ibm_watsonx_ai import APIClient from ibm_watsonx_ai.foundation_models import ModelInference from ibm_watsonx_ai.foundation_models.schema import TextChatParameters  credentials = Credentials(     url = "https://us-south.ml.cloud.ibm.com",     # api_key = "YOUR_API_KEY" # Optional in lab environments ) client = APIClient(credentials)  model_id = "meta-llama/llama-3-2-90b-vision-instruct" project_id = "skills-network" params = TextChatParameters(     temperature=0.2,     top_p=0.6,     max_tokens=2000 )  model = ModelInference(     model_id=model_id,     credentials=credentials,     project_id=project_id,     params=params ) </pre>   |

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

Author

[Hailey Quach](#)



**Skills** Network