**Detailed Beginner’s Guide to Multimodal AI Tasks, Whisper API (Groq & Hugging Face), and Simple Multimodal Apps with LangChain & LlamaIndex**

**Multimodal AI Tasks Explained**

|  |  |  |
| --- | --- | --- |
| Task | What It Means (Beginner Explanation) | Example Use Case |
| **Image Understanding** | AI “looks” at an image and describes or classifies it. | Captioning a photo, detecting objects |
| **Image Generation** | AI creates a new image based on a prompt or another image. | Generating art from text instructions |
| **Video Understanding** | AI analyzes video to recognize actions, objects, or summarize content. | Detecting scenes in a movie |
| **Audio Processing** | AI works with audio signals (not just speech), e.g., music genre classification, noise reduction. | Identifying music genres |
| **Text-to-Speech** | AI converts written text into spoken voice. | Reading articles aloud |
| **Speech-to-Text** | AI transcribes spoken words from audio into written text. | Transcribing interviews |

**Speech-to-Text: Using Whisper API (Groq & Hugging Face)**

**1. Using Groq Whisper API (Fast, OpenAI-Compatible)**

Groq offers extremely fast Whisper-based speech-to-text via an API.  
**Install the SDK:**

pip install groq

**Basic Python Code:**

import os  
from groq import Groq  
  
# Set your Groq API key (set as env variable or directly)  
client = Groq(api\_key=os.getenv("GROQ\_API\_KEY"))  
  
filename = "YOUR\_AUDIO.wav" # Path to your audio file  
  
with open(filename, "rb") as file:  
 transcription = client.audio.transcriptions.create(  
 file=file,  
 model="whisper-large-v3-turbo", # Or "distil-whisper-large-v3-en"  
 response\_format="text", # "text", "json", or "verbose\_json"  
 language="en" # Optional  
 )  
 print(transcription.text)

* **Why Groq?** Fast, accurate, and easy to use for both English and multilingual audio[[1]](#fn1)[[2]](#fn2).

**2. Using Hugging Face Whisper Models (Free, Local, CPU-Friendly)**

**Install dependencies:**

pip install transformers datasets

**Basic Python Code:**

from datasets import load\_dataset  
from transformers import WhisperProcessor, WhisperForConditionalGeneration  
  
# Load a test audio sample  
ds = load\_dataset("hf-internal-testing/librispeech\_asr\_dummy", "clean", split="validation")  
audio\_sample = ds[^0]["audio"]  
  
# Load Whisper model and processor  
processor = WhisperProcessor.from\_pretrained("openai/whisper-tiny.en")  
model = WhisperForConditionalGeneration.from\_pretrained("openai/whisper-tiny.en")  
  
# Preprocess and transcribe  
input\_features = processor(audio\_sample["array"], sampling\_rate=audio\_sample["sampling\_rate"], return\_tensors="pt").input\_features  
predicted\_ids = model.generate(input\_features)  
transcription = processor.batch\_decode(predicted\_ids, skip\_special\_tokens=True)[^0]  
print(transcription)

* **Why Hugging Face?** Free, open-source, runs locally (choose “tiny” or “base” models for CPU)[[3]](#fn3).

**Image Understanding: Captioning with Hugging Face**

**Install dependencies:**

pip install transformers pillow

**Basic Python Code:**

from transformers import AutoProcessor, AutoModelForCausalLM  
from PIL import Image  
import requests  
  
processor = AutoProcessor.from\_pretrained("microsoft/git-base-coco")  
model = AutoModelForCausalLM.from\_pretrained("microsoft/git-base-coco")  
  
url = "http://images.cocodataset.org/val2017/000000039769.jpg"  
image = Image.open(requests.get(url, stream=True).raw)  
  
pixel\_values = processor(images=image, return\_tensors="pt").pixel\_values  
generated\_ids = model.generate(pixel\_values=pixel\_values, max\_length=50)  
caption = processor.batch\_decode(generated\_ids, skip\_special\_tokens=True)  
print(caption)

* **What’s happening?** The model “looks” at the image and generates a descriptive caption[[4]](#fn4).

**Multimodal Apps with LangChain: Passing Images & Text**

LangChain lets you build apps that combine text and images as inputs for chat models

**Example: Send Text + Image to a Multimodal Model**

import base64  
import httpx  
from langchain.chat\_models import init\_chat\_model  
from langchain.schema import HumanMessage  
  
# Download and encode image  
image\_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-wisconsin-madison-the-nature-boardwalk.jpg/2560px-Gfp-wisconsin-madison-the-nature-boardwalk.jpg"  
image\_data = base64.b64encode(httpx.get(image\_url).content).decode("utf-8")  
  
# Compose message with text and image  
message = HumanMessage(content=[  
 {"type": "text", "text": "What do you see in this image?"},  
 {"type": "image", "source\_type": "base64", "mime\_type": "image/jpeg", "data": image\_data}  
])  
  
# Initialize your multimodal chat model (provider-specific) meta-llama/llama-4-scout-17b-16e-instruct  
# model = init\_chat\_model(...) # See provider docs for setup  
  
# response = model.invoke([message])  
# print(response)

* **Note:** Actual model initialization depends on your provider (OpenAI, Gemini, Hugging Face, etc.)[[5]](#fn5)[[6]](#fn6).

**Multimodal with LlamaIndex: Images + Text**

LlamaIndex supports multimodal RAG and image reasoning by combining text and image retrieval[[7]](#fn7)[[8]](#fn8)[[9]](#fn9).

**Image Captioning with Hugging Face via LlamaIndex:**

from llama\_index.multi\_modal\_llms.huggingface import HuggingFaceMultiModal  
from llama\_index.core.schema import ImageDocument  
  
# Initialize model (requires Hugging Face API token)  
model = HuggingFaceMultiModal.from\_model\_name("Qwen/Qwen2-VL-2B-Instruct")  
image\_document = ImageDocument(image\_path="path/to/your/image.jpg")  
prompt = "Describe this image in detail."  
  
response = model.complete(prompt, image\_documents=[image\_document])  
print(response.text)

* **Why LlamaIndex?** Lets you combine text and image retrieval, and use multimodal LLMs for answering questions or generating captions[[7]](#fn7)[[9]](#fn9)[[8]](#fn8).

**Building a Simple Multimodal App: Example Workflow**

1. **User uploads an image or audio.**
2. **App transcribes audio (Whisper) or captions image (GIT-base or similar).**
3. **App displays the result or sends it to a chat model for further Q&A.**

**Example: Multimodal Q&A with Image and Text**

**With LangChain (Conceptual):**

# Prepare your multimodal message  
message = HumanMessage(content=[  
 {"type": "text", "text": "What is happening in this photo?"},  
 {"type": "image\_url", "image\_url": {"url": "https://example.com/photo.jpg"}}  
])  
  
# Send to a multimodal-capable chat model  
# response = model.invoke([message])  
# print(response)

* **Note:** Actual model support and initialization varies by provider[[5]](#fn5)[[6]](#fn6).

**Key Points for Beginners**

* **Image Understanding:** Captioning, classification (use Hugging Face or LlamaIndex).
* **Image Generation:** Use models like Stable Diffusion (not shown here, but similar setup).
* **Audio/Speech-to-Text:** Use Whisper via Groq API or Hugging Face.
* **Multimodal Apps:** Use LangChain or LlamaIndex to combine text, images, and audio for richer user experiences.
* **Choose lightweight models for local use:** e.g., whisper-tiny.en for CPU, or free Hugging Face models for image tasks.

**You can start with these code snippets, swap out models as needed, and gradually build more advanced multimodal applications as you get comfortable with the basics.**

#groq multimodal

import base64

def encode\_image(image\_path):

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

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

import os

from groq import Groq

client = Groq(api\_key=os.environ.get("GROQ\_API\_KEY"))

# Encode your local image

image\_path = "your\_image.jpg"

base64\_image = encode\_image(image\_path)

# Compose the message with text and image

messages = [

{

"role": "user",

"content": [

{"type": "text", "text": "What do you see in this image?"},

{

"type": "image\_url",

"image\_url": {

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

}

}

]

}

]

# Call the Groq multimodal model

completion = client.chat.completions.create(

model="meta-llama/llama-4-scout-17b-16e-instruct",

messages=messages,

max\_completion\_tokens=1024,

temperature=1

)

print(completion.choices[0].message.content)