# Image Captioning Project Report

## 1. Project Overview & Goals

Objective: Build an end-to-end image captioning application that leverages the BLIP API.

Components:

• API Integration: A function to send an image to the BLIP API and receive a generated caption.

• Interactive Interface: Using Gradio to create a web interface that allows users to upload images and view captions.

• Security: Best practices for managing API keys by using a .env file.

## 2. Detailed Code Explanation

### Importing Libraries and Environment Setup

• from dotenv import load\_dotenv: Loads the environment variables from a file named .env into your Python environment.

• import os: Provides a portable way of using operating system–dependent functionality (e.g., retrieving environment variables).

• import requests: Allows sending HTTP requests to the BLIP API.

• import gradio as gr: Used to build the interactive web interface.

• import io: Provides Python’s I/O stream capabilities (used here for in-memory binary streams).

• from PIL import Image: Imports the Python Imaging Library (Pillow) for image processing.

• import numpy as np: Allows array manipulations (useful for handling images in different formats).

### Environment Variable Loading

• load\_dotenv(): Reads the .env file and loads the variables.

• API\_KEY = os.getenv("HF\_API\_KEY"): Retrieves the Hugging Face API key from the environment.

### API Configuration

• API\_URL = "https://api-inference.huggingface.co/models/Salesforce/blip-image-captioning-base": Defines the BLIP image captioning API endpoint.

• headers = {"Authorization": f"Bearer {API\_KEY}"}: Sets the HTTP header to include the API key securely in the request.

### Function: generate\_caption(image)

#### Input Handling:

• If the input is a file path (string), it opens the image using Image.open(image).

• If the input is a NumPy array, it converts the array to an image using Image.fromarray(image).

#### Image Conversion:

• image = image.convert("RGB"): Ensures the image is in RGB mode (necessary for JPEG conversion).

#### Image Serialization:

• A BytesIO buffer is created to hold the binary image data.

• image.save(buf, format="JPEG"): Saves the image to the buffer in JPEG format.

• data = buf.getvalue(): Retrieves the binary data from the buffer.

#### API Request:

• response = requests.post(API\_URL, headers=headers, data=data): Sends the image binary data to the BLIP API.

• result = response.json(): Converts the JSON response into a Python dictionary.

#### Error Handling & Output:

• Checks if the response contains an error; if so, returns the error message.

• Otherwise, it returns the generated caption from the API result.

### Gradio Interface Setup

• iface = gr.Interface(...): Creates an interactive interface with user input as an image and output as text.

• fn=generate\_caption: The function to call when an image is uploaded.

• inputs=gr.Image(type="pil"): Specifies that the input is an image (automatically converted to a PIL image).

• outputs="text": The output will be displayed as text.

• title and description: Provide context and instructions for the user.

• iface.launch(share=True): Launches the interface. The share=True flag generates a public URL (if possible) for sharing.

## 3. Decision on Training from Scratch

We initially explored the idea of training a model on the Flickr dataset.

### Observation:

• Training on 8091 images was estimated to take around 42 hours on our available hardware.

• Even a reduced dataset of 2500 images required around 11 hours.

### Decision:

Given the computational time and resources required, we decided to use the BLIP API (a zero-training approach) instead, which allows us to focus on building, deploying, and integrating the solution without the heavy training overhead.