Setup

This first cell installs all the necessary libraries for this notebook. We are using `langchain`, `gradio`, `transformers`, `bs4`, `requests`, and `torch`.

```
# installing required libraries in my_env
!pip install langchain==0.1.11 gradio==5.23.2 transformers==4.38.2 bs4==0.0.2 requests==2.31.0 torch==2.2.1
```

1- Image Captioning Model

Here we are importing the necessary libraries for the image captioning part of the notebook: `requests` to download images from the web, `PIL` (Pillow) to work with images, and `transformers` from Hugging Face to use a pre-trained image captioning model.

```
import requests
from PIL import Image
from transformers import AutoProcessor, BlipForConditionalGeneration

# Load the pretrained processor and model
processor = AutoProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart you
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
preprocessor_config.json: 100%  287/287 [00:00<00:00, 20.7kB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default behavior in v4.52, even if the model w
tokenizer_config.json: 100%  506/506 [00:00<00:00, 30.5kB/s]
vocab.txt: 232k/? [00:00<00:00, 10.3MB/s]
tokenizer.json: 711k/? [00:00<00:00, 40.3MB/s]
special_tokens_map.json: 100%  125/125 [00:00<00:00, 12.9kB/s]
config.json: 4.56k/? [00:00<00:00, 437kB/s]
pytorch_model.bin: 100%  990M/990M [00:07<00:00, 85.9MB/s]
```

This cell loads an image from a specified path. The code then converts the image to the RGB format, which is required by the image captioning model.

```
img_path = "/content/sun.png"

# convert it into an RGB format
image = Image.open(img_path).convert('RGB')

```

```
/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

Here, we prepare the image and an optional text input for the model. For image captioning, a generic text like "the image about what" is used as a prompt. The `processor` converts the image and text into the format that the model expects.

```
# You do not need a question for image captioning
text = "This image shows"
inputs = processor(images=image, text=text, return_tensors="pt")
```

The `generate` method of the model produces a sequence of tokens representing the caption. We limit the caption length to 50 tokens using `max_length=50`.

```
# Generate a caption for the image
outputs = model.generate(**inputs, max_length=50)
```

After generating the caption tokens, this cell decodes them back into a human-readable text string. The `processor.decode` method is used for this purpose, and `skip_special_tokens=True` removes any special tokens added during the generation process.

```
# Decode the generated tokens to text
caption = processor.decode(outputs[0], skip_special_tokens=True)
# Print the caption
print(caption)
```

```
this image shows the sun with a smiley face
```

2- Gradio Interface

Import the necessary libraries for building the Gradio interface for our image captioning model.

```
import gradio as gr
import numpy as np
from PIL import Image
from transformers import AutoProcessor, BlipForConditionalGeneration
```

This function `caption_image` is the core logic for the Gradio interface. It takes a NumPy array representing an image as input. It converts the image to the correct format, uses the processor to prepare the input for the model, generates the caption using the model, and finally decodes the output into a readable string. This string is then returned as the output of the function.

```
def caption_image(input_image: np.ndarray):
    # Convert numpy array to PIL Image and convert to RGB
    raw_image = Image.fromarray(input_image).convert('RGB')

    # Process the image
    inputs = processor(raw_image, return_tensors="pt")

    # Generate a caption for the image
    outputs = model.generate(**inputs)

    # Decode the generated tokens to text and store it into `caption`
    caption = processor.decode(outputs[0], skip_special_tokens=True)

    return caption
```

This cell defines the Gradio interface for the image captioning task. It uses the `caption_image` function as the core logic. The interface has an image input (`gr.Image()`) and a text output (`"text"`). We also provide a title and description for the web app.

```
iface = gr.Interface(
    fn=caption_image,
    inputs=gr.Image(),
    outputs="text",
    title="Image Captioning",
    description="This is a simple web app for generating captions for images using a trained model."
)
```

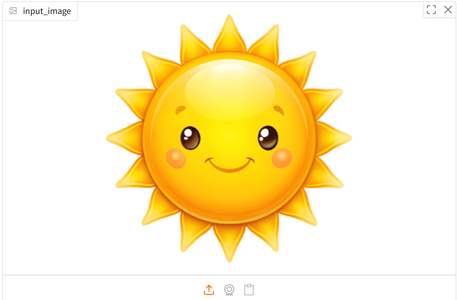
Finally, this cell launches the Gradio interface. Once executed, it will provide a public URL that you can use to access the image captioning web app.

```
iface.launch()
```

```
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you can turn this off by setting `share=Fa

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://dfa1efea24afb23025.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Space
```



Next steps: Deploy to Cloud Run