Approach and Experiments:

Based on the project's background, I opted to focus on a solution that could handle a wide variety of images, as our clients' needs can significantly differ depending on their business domains. I decided to use a pre-trained model, recognizing that training or fine-tuning such models demands substantial data and computing resources, which I lacked. Consequently, I did not create a custom dataset.

For model selection, I referred to HuggingFace's models page, concentrating on Visual Transformers paired with a Language Model decoder like GPT2. I initially focused on models designed to generate image captions and experimented with the following models:

- **Vit-gpt2-image-captioning**
- **Blip-image-captioning-large**
- **Git-base-coco**

I selected these models based on their popularity for image captioning and tested them on a few downloaded images. While the captions were good, they were very concise, often providing only brief sentences rather than detailed descriptions. I tried increasing the number of min_tokens and the number of beams to generate longer captions. However, after a few tokens, the models tended to repeat the same words instead of producing logically longer text. This led me to conclude that conciseness is what these models were trained for.

After spending a significant amount of time searching, I couldn't find models specifically designed for **detailed image descriptions**. However, I realized that we could generate detailed descriptions by using a prompt or question. This led me to explore **visual-question-answering** models.

- **MiniCPM - LLama 3** and **Mini-InternVL-Chat-4B.** I experimented with these models using the HuggingFace GUIs and found that they provided very accurate and detailed descriptions. However, I realized that these models are too large, ranging from 5 to 12 billion parameters, making them impractical for local use on my CPU. As an M1 Mac user, utilizing my GPU is not very feasible, which further motivated my search for smaller, more efficient models that can run on a CPU without significant performance drawbacks.
- **Moondream2,** a smaller model with 1.87 billion parameters, which provided very good descriptions. However, I had to modify some code to run it on my CPU, and even then, it took over 15 minutes to generate a caption, so I decided not to use it further. Using Google Colab reduced the time to just 2 minutes, but I noticed that the model required more than 10 GB of RAM, while my PC only had 8 GB.
- **Deepseek-vl-1.3b-chat** which had 1.3B params, but on my local machine took 6-7 minutes for prediction, which I still found not good enough.

These were the large models that I tried. Next, I experimented with a couple of smaller **visual-question-answering** models like **Vilt-b32-finetuned-vqa** and **Blip-vqa**, which had 0.3-0.5 billion parameters. However, they only provided very short and concise answers instead of detailed descriptions.

Finally, I found and employed **TinyLLaVA-OpenELM**, which has 0.89 billion parameters and successfully runs on my machine. It can generate **detailed descriptions** for images, with an inference time of 2-3 minutes per image. The drawback is the quality: while it captures the essence of the image, it can also hallucinate. For example, when I input an image of a

playground, it described it in detail but mentioned that children were playing, which wasn't true. **Reducing the temperature** improved this by making the model **less random** in its predictions. Despite its flaws, **it fulfils the purpose** of generating **detailed image descriptions** that are good enough for **search and categorization.**

I experimented with changing parameters like **temperature**, **max_length**, and **num_beams** for all the models. For example, I used **visual-question-answering** models to answer the question "What are the main colours?" with a lower temperature than when generating a description, as the specific question required the model to be even less random in its predictions.

Final Setup:

In the end, I created 2 setups:

1. **Normal -** I employed the **TinyLLaVA-OpenELM** model to generate detailed image descriptions. Additionally, I combined this detailed description with an output that describes the main colours of the image. This approach showcases the flexibility of the provided solution, demonstrating that we can easily change the output format. Each inference takes 2-3 minutes.
2. **Fast -** This solution is faster than the **Normal** one but generates a caption instead of a detailed description. I use the **Blip-image-captioning-large** model to generate the caption. Then, I use the same model with a conditional prompt to generate the main colours of the image. Finally, I combined the caption and the colour output. Each inference takes 40-55 seconds.

I have created a folder and downloaded some distinct images and evaluated each model on these images and saved the generated output which you can find in **images/fast_descriptions** and **images/normal_descriptions.**

Usage:

For usage, I have created:

1. A simple Flask web app that exposes one API route. After activating the local Flask server, we can generate descriptions using the CLI. Technically, you can use the CLI to directly get inferences from the models without involving a Flask server. However, I **intentionally designed** it this way to demonstrate the communication between a local script and exposed API routes.
2. A simple Streamlit application, which serves as a graphical user interface (GUI). This app allows you to upload images, visualize them, and generate descriptions directly within the application.

For detailed instructions, please refer to the README.md

Next Steps:

1. I would like to utilize a much better model, specifically **MiniCPM-Llama3-V-2_5**, which I believe is the current **state-of-the-art** for this task. I want to experiment with running a quantized version of the model on my CPU to see if I can achieve faster

inference. Additionally, I want to deploy the model and connect it to a GPU resource in the cloud to ensure optimal performance and faster processing times. This approach will allow us to leverage the enhanced computational power of GPUs to handle the model's demands effectively.

2. I want to get more familiar with the numerous metrics that define performance. For example, I saw the GQA, which refers to the accuracy in the Generalized Question Answering dataset and corresponds to our scenario. Understanding these metrics will help in accurately evaluating and comparing the performance of various models. However, it is worth noting that MiniCPM-Llama3-V-2_5 has not been evaluated using the GQA dataset yet. Nonetheless, gaining a deeper understanding of these performance metrics will be beneficial.
3. If I have access to some of the images that our clients would want to search or categorize, I plan to test our model on them. This will provide practical insights into how well the model performs in real-world scenarios and ensure that it meets the specific needs of our clients. Testing on client-specific images will help us fine-tune the model to deliver more accurate and relevant results.
4. An open-source model like **MiniCPM-Llama3-V-2_5** may have problems with **hallucinations** and **data leakage**. I would also like to research this area to understand the potential risks and develop strategies to mitigate them. Addressing these issues is crucial to ensuring the reliability and trustworthiness of the model's outputs. This research will involve studying current literature, exploring best practices, and potentially implementing safeguards within our model deployment process.
5. Finally, my goal is to develop an end-to-end product, deploy it in production, and monitor its performance. This involves integrating the model into a user-friendly interface, optimizing it for the chosen infrastructure, and implementing tools to track performance and detect issues. By doing so, I aim to create a scalable solution that provides reliable and accurate image descriptions for our clients.