### **Project Documentation: AI Image Captioning App: Describe Your Images with AI with Gemini Vision Pro**

**Version:** 1.0.0

**Date:** June 19, 2025

**Author(s):** Saumya Dwivedi (23BCE10331), Sakshi Mittal (23BCE11231), Shriyash Joshi (23BCE11284)

**Contact:** saumya.23bce10331@vitbhopl.ac.in, sakshi.23bce11231@vitbhopal.ac.in, shriyash.23bce11284@vitbhopal.ac.in

**1. Introduction**

In today's digital landscape, visual content reigns supreme, especially on social media. For social media managers, content creators, and businesses, generating engaging and relevant captions for images is crucial but often time-consuming. The "AI Image Captioning App" leverages the power of Google's Gemini Vision Pro model to automate this process, providing instant, suitable captions for uploaded images. This application aims to streamline content creation workflows, allowing users to focus more on strategy and less on repetitive caption writing.

### **2.** Problem Definition & Use Case

**Problem:** Manual image captioning is a tedious and time-consuming task, especially for individuals or organizations dealing with a large influx of images, such as social media managers, content creators, or e-commerce businesses. This manual process can lead to inconsistencies in caption quality, delays in content deployment, and a significant drain on resources.

**3. Use Case: Photo Captioning**

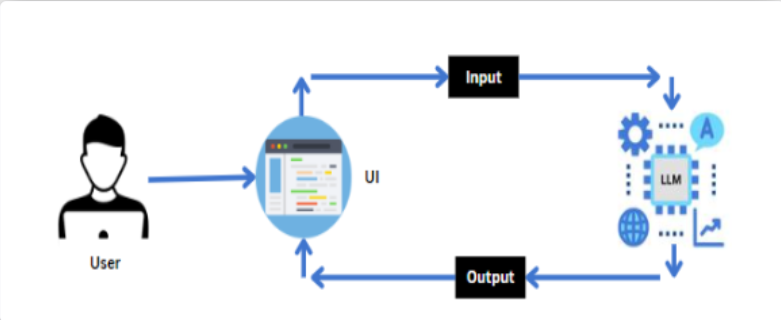
Imagine you are a social media manager for a popular brand. Each day, you need to create engaging posts that include eye-catching images and compelling captions. Crafting unique captions for each image can be a time-consuming task, especially when managing multiple social media accounts. You need a solution that can help you quickly generate suitable captions for the images you want to post. This AI Image Captioning App directly addresses this need by providing an efficient and automated way to generate relevant captions, allowing the social media manager to focus on strategy and engagement rather than repetitive caption writing.

### 4.Generative AI Model & Architecture

The core of this AI Image Captioning App is built upon the Gemini Pro pre-trained model, a state-of-the-art Generative AI model from Google. The architecture follows a client-server model:

* **User Interface (UI):** Developed using Streamlit, providing an intuitive web interface for users to upload images and trigger caption generation.
* **Backend:** Handles the communication with the Gemini Pro model. User input (image and optional text prompt) is collected from the UI and transmitted to the backend using the Google API Key.
* **Gemini Pro Pre-trained Model:** Receives the input via an API call, processes the image and associated prompt, and generates a suitable textual caption. The model utilized is gemini-1.5-flash.
* **Output:** The generated caption is returned from the backend to the frontend for formatting and display to the user.

**Architecture Diagram:**



### 5. Data Management

For this project, the primary "data" being managed is the user-uploaded image.

* **Input Data:** Images uploaded by the user through the Streamlit UI (formats supported: JPG, JPEG, PNG). These images are temporarily processed in memory as byte data before being sent to the Gemini API.
* **Output Data:** The generated textual captions from the Gemini Pro model.

There is no persistent storage of user-uploaded images or generated captions within the application itself. The application processes the data on demand, focusing on real-time caption generation.

### 6. Training and Experimentation

As this project leverages a pre-trained model (Gemini Pro), no explicit training or experimentation phase is required by the user for the model itself. The Gemini Pro model has been extensively pre-trained by Google on a vast dataset.

The "experimentation" in this context would involve:

* **Prompt Engineering:** Experimenting with different input\_prompt variations to fine-tune the quality and style of the generated captions. The current input\_prompt is set to: "You are an expert image captioning so whatever image you will get give a suitable caption in one phrase." Further refinements to this prompt could explore different lengths, tones, or specific requirements for captions.
* **Error Handling:** Thorough testing of the implemented error handling for various API exceptions (e.g., InvalidArgument, FailedPrecondition, ResourceExhausted, DeadlineExceeded) to ensure a robust user experience.

### 7. Evaluation and Metrics

Given the nature of generative AI for captioning, quantitative metrics can be challenging to define definitively without a ground truth dataset. However, evaluation can be conducted through:

* **Qualitative Assessment:** The primary evaluation metric will be the human assessment of the generated captions' relevance, accuracy, conciseness, and suitability for the given image and use case.
* **User Feedback:** Gathering feedback from target users (e.g., social media managers) on the utility and effectiveness of the generated captions in their workflow.
* **Error Rate Monitoring:** Tracking the frequency of API errors encountered to identify potential issues with API key limits, network connectivity, or model availability.

### 8. Ethical Considerations & Responsible AI

Developing an AI image captioning application necessitates careful consideration of ethical implications and responsible AI practices:

* **Bias in Captions:** Pre-trained models can inherit biases present in their training data. It's crucial to be aware that the generated captions might occasionally reflect societal biases or stereotypes. Regular monitoring and potentially implementing content moderation or user reporting mechanisms could mitigate this.
* **Misinterpretation/Misinformation:** While the goal is accurate captions, the model might occasionally misinterpret an image, leading to inaccurate or misleading captions. It's important to clearly state that the AI-generated captions are suggestions and should be reviewed by a human.
* **Privacy:** The application should not store or misuse user-uploaded images. Emphasize that images are processed transiently for caption generation.
* **Transparency:** Users should be informed that the captions are AI-generated.
* **Harmful Content:** Implement measures to prevent the generation of captions for, or descriptive of, harmful, offensive, or inappropriate content. While the Gemini API likely has built-in safety filters, an additional layer of content filtering might be considered for sensitive applications.

### 9. Future Work & Enhancements

* **Caption Style Customization:** Allow users to specify desired caption styles (e.g., humorous, professional, poetic, marketing-focused).
* **Multi-language Support:** Extend caption generation to multiple languages.
* **Integration with Social Media Platforms:** Direct integration with popular social media platforms for seamless posting of images with generated captions.
* **User Feedback Loop for Model Improvement:** Implement a mechanism for users to provide feedback on caption quality, which could potentially be used to fine-tune future model iterations or improve prompt engineering.
* **Batch Processing:** Enable users to upload multiple images at once and generate captions for all of them.
* **Caption Editing:** Provide an in-app editor for users to easily modify or refine the generated captions.
* **Deployment to Cloud Platforms:** Host the application on scalable cloud platforms like Google Cloud Run or AWS Elastic Beanstalk for production-level deployment.

### 10. Version Control & Repository

* **Code Repository:** https://github.com/mittalsaks/Gen\_Ai-.git
* **Branching Strategy:** A simple branching strategy is employed for this project to ensure a clear development workflow. The main branch will host the stable, production-ready code. The develop branch will be used for active development and integration of new features. For individual new features or significant bug fixes, dedicated feature branches will be created, branching off from develop and merging back upon completion and review.
* **Version History (of documentation):**
  + v1.0.0 (June 19, 2025): Initial release of project documentation.

## 11. How to Use / Getting Started

This section outlines how to set up and run the AI Image Captioning App locally.

### 11.1 Project Structure:

To ensure proper functioning, your project folder should contain the following files:

* .env: This file securely stores your Google AI Studio API key. It is crucial for authenticating with the Gemini Pro model.
* app.py: This is the main application file. It contains both the Streamlit UI code for the web interface and the Python logic for interacting with the Gemini Pro model.
* requirements.txt: This file lists all the necessary Python libraries that need to be installed for the application to run correctly.

### 11.2 Requirements File (requirements.txt):

Create a file named requirements.txt in your project folder with the following content. These are the core libraries required for the application:

streamlitgoogle-generativeaipython-dotenvPillowgoogle-api-core

**Note:**google-api-core has been added to the requirements.txt as it's used for handling API exceptions in your app.py code.

### 11.3 Installation Guide:

Follow these steps to install the necessary libraries:

1. **Open your terminal or Anaconda Prompt.**
2. **Navigate to your project directory** using the cd command (e.g., cd path/to/your/project\_folder).
3. **Install the required libraries** by running the following command:
4. Bash
5. pip install -r requirements.txt

### 11.4 Initialization of Google API Key:

To connect to the Gemini Pro model, you need a Google API Key.

1. **Generate a Google API Key:**
   * Go to the Google AI Studio documentation: <https://ai.google.dev/gemini-api/docs/api-key>
   * Sign in with your Google account.
   * Navigate to 'Get an API Key'.
   * Click 'Create API Key' and choose 'Generative Language Client' as the project. Select 'Create API key in existing project'.
   * Copy the newly generated API key.
2. **Initialize Google API Key (.env file):**
   * In your project folder, create a new file named .env.
   * Open the .env file and add the following line, replacing your\_google\_api\_key\_here with the API key you copied:
   * GOOGLE\_API\_KEY="your\_google\_api\_key\_here"
   * **Important:** Never share your .env file or commit it to public version control repositories (like GitHub, GitLab, Bitbucket). This file contains sensitive credentials.

### 11.5 Code Structure (app.py):

The app.py file contains the Python code for your Streamlit application, handling the UI and interaction with the Gemini model. Your provided code snippet is used directly for this section.

**Python**

import streamlit as st

import os

import google.generativeai as genai

from PIL import Image

from dotenv import load\_dotenv

import google.api\_core.exceptions # Import the exceptions module

# Load environment variables

load\_dotenv()

# Configure Generative AI with API key

# Ensure GOOGLE\_API\_KEY is set in your .env file

api\_key = os.getenv("GOOGLE\_API\_KEY")

if not api\_key:

st.error("Google API Key not found. Please set the GOOGLE\_API\_KEY environment variable in your .env file.")

st.stop() # Stop the app if API key is missing

genai.configure(api\_key=api\_key)

# Function to get Gemini response

def get\_gemini\_repsonse(input\_text, image\_parts, prompt):

try:

model = genai.GenerativeModel('gemini-1.5-flash')

# Ensure image\_parts is a list of dictionaries as expected by the model

response = model.generate\_content([input\_text, image\_parts[0], prompt])

return response.text

except google.api\_core.exceptions.InvalidArgument as e:

st.error(f"Invalid argument error from Gemini API: {e}. This might indicate an issue with the prompt or image data.")

return None

except google.api\_core.exceptions.FailedPrecondition as e:

st.error(f"Failed precondition error from Gemini API: {e}. This can happen if the model is not available or if there's an issue with the request setup.")

return None

except google.api\_core.exceptions.ResourceExhausted as e:

st.error(f"Quota exceeded error from Gemini API: {e}. You might have hit your usage limits. Please check your Google Cloud Console for details.")

return None

except google.api\_core.exceptions.DeadlineExceeded as e:

st.error(f"API call timed out: {e}. This might be due to network issues or slow model response.")

return None

except google.api\_core.exceptions.GoogleAPICallError as e:

st.error(f"An error occurred with the Gemini API call: {e}. Please check your API key and network connection.")

return None

except Exception as e:

st.error(f"An unexpected error occurred: {e}. Please try again.")

return None

# Function to prepare image for Gemini

def input\_image\_setup(uploaded\_file):

if uploaded\_file is not None:

bytes\_data = uploaded\_file.getvalue()

image\_parts = [

{

"mime\_type": uploaded\_file.type,

"data": bytes\_data

}

]

return image\_parts

else:

raise FileNotFoundError("No file uploaded")

# Define the input prompt for image captioning

input\_prompt = """You are an expert image captioning so whatever image you will get give a suitable caption in one phrase."""

# Streamlit UI

st.set\_page\_config(page\_title="IMAGE CAPTIONING")

st.header("IMAGE TO CAPTION")

# Input prompt from user (can be left empty if not used for the specific task)

user\_input\_text = st.text\_input("Additional Context/Prompt (Optional):", key="input")

# Image uploader

uploaded\_file = st.file\_uploader("Choose an image...", type=["jpg", "jpeg", "png"])

image = None # Initialize image to None

if uploaded\_file is not None:

image = Image.open(uploaded\_file)

st.image(image, caption="Uploaded Image.", use\_column\_width=True)

# Submit button

submit = st.button("Generate Caption")

# When submit button is clicked

if submit:

if uploaded\_file is not None:

try:

image\_data = input\_image\_setup(uploaded\_file)

# Pass the user\_input\_text as the 'input' argument to the model.

# The main instruction for captioning comes from input\_prompt.

response\_text = get\_gemini\_repsonse(user\_input\_text, image\_data, input\_prompt)

if response\_text:

st.subheader("The Caption Is:")

st.write(response\_text)

except FileNotFoundError as e:

st.error(str(e))

except Exception as e:

st.error(f"An error occurred during image processing: {e}")

else:

st.warning("Please upload an image first to generate a caption!")

### 11.6 Running the Web Application:

Once you have set up your project folder, installed the dependencies, and configured your API key, you can run the application:

1. **Open your terminal or Anaconda Prompt.**
2. **Navigate to your project folder.**
3. **Run the Streamlit application** using the command:
4. Bash
5. streamlit run app.py
6. Your default web browser should automatically open to the Streamlit app (usually at **http://localhost:8501**). If it does not, manually navigate to that address in your web browser.

## 12. References & Acknowledgements

This project was built upon the foundation of several excellent resources and open-source tools.

### 12.1 Key Resources:

* **Google Gemini API Documentation:**<https://ai.google.dev/gemini-api>
* **Streamlit Documentation:**<https://docs.streamlit.io/>

### 12.2 Libraries Used:

* Streamlit
* Google Generative AI Python SDK
* Python-dotenv
* Pillow (PIL)
* Google API Core