Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Subject: NLP Branch: AIML

Name	**Jash Vora**
UID	**2022600065**
Batch	**A1**
Course	**NLP**
Exp no.	**7**
Name of the Experiment	Building Agentic AI Projects with Optional Streamlit Frontend
AIM	To design, develop, and deploy an Agentic AI system that autonomously perceives, plans, and acts to achieve a goal, with an optional web-based interface using Streamlit.
Theory	Steps / Procedure 1. Define Problem Statement ○ Select an agent idea. ○ Frame clear goals and success metrics. 2. System Design ○ Identify the perception layer (inputs). ○ Plan reasoning logic (rules, LLM, RL). ○ Define the action layer (outputs, tasks). 3. Model Development ○ Use LLM APIs or train ML models.

	○ Add memory if the agent needs context. 4. Integration ○ Connect to APIs (search engines, databases, video/audio tools). ○ For RL: set up environment, states, rewards. 5. Frontend (Optional) ○ Build a simple Streamlit dashboard: ■ Input fields for prompts/queries ■ Display agent outputs & logs 6. Testing & Debugging ○ Check accuracy, reliability, and responsiveness. 7. Documentation ○ Write clear usage steps, screenshots, and limitations.
Code	

```
import os
import base64
from pathlib import Path
from typing import Optional, Union
import requests
from dotenv import load_dotenv
from PIL import Image
import io

# Load environment variables
load_dotenv()

class GeminiImageCaptionAgent:
    def __init__(self, api_key: Optional[str] = None):
        """
        Initialize the Gemini Image Caption Agent

        Args:
                    api_key: Gemini API key. If None, will load from
GEMINI_API_KEY env var
```

```
        """
        self.api_key = api_key or os.getenv('GOOGLE_GEMINI_API_KEY')
        if not self.api_key:
                    raise ValueError("Gemini API key not found. Set
GEMINI_API_KEY in .env or pass as parameter")

                                                self.base_url      =
"https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-fl
ash:generateContent"

         def _encode_image(self, image_path: Union[str, Path]) ->
tuple[str, str]:
        """
        Encode image to base64 and detect mime type

        Args:
            image_path: Path to the image file

        Returns:
            Tuple of (base64_string, mime_type)
        """
        image_path = Path(image_path)
        if not image_path.exists():
                     raise FileNotFoundError(f"Image file not found:
{image_path}")

        # Open and validate image
        try:
            with Image.open(image_path) as img:
                  # Convert to RGB if necessary (handles RGBA, P mode,
etc.)
                if img.mode != 'RGB':
                    img = img.convert('RGB')

                # Convert to bytes
                buffer = io.BytesIO()
                img.save(buffer, format='JPEG', quality=95)
                image_bytes = buffer.getvalue()
```

```
        except Exception as e:
            raise ValueError(f"Invalid image file: {e}")

        # Encode to base64
        base64_string = base64.b64encode(image_bytes).decode('utf-8')

        return base64_string, "image/jpeg"

    def generate_caption(self,
                         image_path: Union[str, Path],
                                 prompt: str = "Generate a detailed and
accurate caption for this image.",
                         max_tokens: int = 150,
                         temperature: float = 0.7) -> str:
        """
        Generate a caption for the given image

        Args:
            image_path: Path to the image file
            prompt: Custom prompt for caption generation
            max_tokens: Maximum number of tokens in response
            temperature: Sampling temperature (0.0 to 1.0)

        Returns:
            Generated caption as string
        """
        try:
            # Encode image
            base64_image, mime_type = self._encode_image(image_path)

            # Prepare request payload
            payload = {
                "contents": [
                    {
                        "parts": [
                            {"text": prompt},
                            {
                                "inline_data": {
                                    "mime_type": mime_type,
```

```
                                    "data": base64_image
                                }
                            }
                        ]
                    }
                ],
                "generationConfig": {
                    "maxOutputTokens": max_tokens,
                    "temperature": temperature
                }
            }

            # Make request
            response = requests.post(
                f"{self.base_url}?key={self.api_key}",
                json=payload,
                headers={"Content-Type": "application/json"}
            )

            if response.status_code != 200:
                                raise  Exception(f"API  request  failed:
{response.status_code} - {response.text}")

            result = response.json()

            # Extract caption from response
             if 'candidates' in result and len(result['candidates']) >
0:
                candidate = result['candidates'][0]
                        if  'content'  in  candidate  and  'parts'  in
candidate['content']:
                                                                caption   =
candidate['content']['parts'][0]['text'].strip()
                    return caption
                else:
                        raise Exception("Unexpected response format:
missing content")
            else:
                raise Exception("No caption generated")
```

```
        except Exception as e:
            raise Exception(f"Error generating caption: {str(e)}")

    def generate_detailed_caption(self, image_path: Union[str, Path])
-> str:
        """
        Generate a detailed caption with specific prompting

        Args:
            image_path: Path to the image file

        Returns:
            Detailed caption as string
        """
        detailed_prompt = """
          Analyze this image carefully and provide a detailed caption
that includes:
        1. Main subjects and objects in the image
        2. Setting and environment
        3. Colors, lighting, and mood
        4. Actions or activities taking place
        5. Any notable details or interesting elements

        Write the caption in a natural, descriptive style.
        """

            return self.generate_caption(image_path, detailed_prompt,
max_tokens=200)

    def generate_simple_caption(self, image_path: Union[str, Path]) ->
str:
        """
        Generate a simple, concise caption

        Args:
            image_path: Path to the image file

        Returns:
```

```
            Simple caption as string
        """
           simple_prompt = "Describe this image in one clear, concise
sentence."

              return  self.generate_caption(image_path,  simple_prompt,
max_tokens=50)

    def batch_caption_images(self, image_folder: Union[str, Path],
                             output_file: Optional[str] = None) -> dict:
        """
        Generate captions for all images in a folder

        Args:
            image_folder: Path to folder containing images
            output_file: Optional file to save results

        Returns:
            Dictionary mapping image filenames to captions
        """
        image_folder = Path(image_folder)
        if not image_folder.exists():
                    raise  FileNotFoundError(f"Image  folder  not  found:
{image_folder}")

        # Supported image extensions
            supported_extensions  =  {'.jpg',  '.jpeg',  '.png',  '.bmp',
'.gif', '.tiff'}

        results = {}
        image_files = [f for f in image_folder.iterdir()
                       if f.suffix.lower() in supported_extensions]

        print(f"Processing {len(image_files)} images...")

        for i, image_file in enumerate(image_files, 1):
            try:
                             print(f"Processing  {i}/{len(image_files)}:
{image_file.name}")
```

```
                caption = self.generate_caption(image_file)
                results[image_file.name] = caption

            except Exception as e:
                print(f"Error processing {image_file.name}: {e}")
                results[image_file.name] = f"Error: {str(e)}"

        # Save results if output file specified
        if output_file:
            with open(output_file, 'w', encoding='utf-8') as f:
                for filename, caption in results.items():
                    f.write(f"{filename}: {caption}\n\n")
            print(f"Results saved to {output_file}")

        return results


def main():
    """
    Example usage of the Gemini Image Caption Agent
    """
    try:
        # Initialize the agent
        agent = GeminiImageCaptionAgent()

        # Example 1: Single image caption
           image_path = "Screenshot 2025-01-24 231045.png"   # Replace
with your image path

        print("Generating simple caption...")
        simple_caption = agent.generate_simple_caption(image_path)
        print(f"Simple Caption: {simple_caption}")

        print("\nGenerating detailed caption...")
        detailed_caption = agent.generate_detailed_caption(image_path)
        print(f"Detailed Caption: {detailed_caption}")

        # Example 2: Custom prompt
```

```
        custom_prompt = "Describe this image as if you were writing alt text for accessibility."
                custom_caption = agent.generate_caption(image_path, custom_prompt)
        print(f"\nCustom Caption: {custom_caption}")

        # Example 3: Batch processing (uncomment to use)
                # results = agent.batch_caption_images("./images", "captions_output.txt")
        # print(f"\nProcessed {len(results)} images")

    except Exception as e:
        print(f"Error: {e}")


if __name__ == "__main__":
    main()
```

Output

```
Generating simple caption...
Simple Caption: The image shows three cars, a Hyundai Elantra, an AM General Hummer, and a Toyota Sequoia, with accurate predictio
ns of their make, model, and year.

Generating detailed caption...
Detailed Caption: Here's a caption describing the image:

This image shows three different vehicles, each presented as a close-up front view.  The first is a silver 2007 Hyundai Elantra se
dan, parked in what appears to be a car dealership or outdoor lot, with a muted gray background and a small American flag visible
in the upper right corner.  The second is a dark olive-green 2000 AM General Hummer SUV, set against a slightly blurred backdrop o
f other vehicles and a hint of a desert-like landscape. The third is a silver 2012 Toyota Sequoia SUV, also in a parking lot with
a less-defined background, possibly a city street.  The lighting in all three images is bright and natural, likely daylight condit
ions, creating a clear and well-lit presentation of each vehicle. The overall mood is neutral and informative, possibly suggesting
 a catalog or database entry for vehicle identification or classification.  Each vehicle is clearly labeled with its

Custom Caption: Here's alt text for the image:

"Three images showing the accuracy of a vehicle identification system.  The left image shows a silver 2007 Hyundai Elantra sedan,
correctly identified.  The center image shows a dark green 2000 AM General Hummer SUV, also correctly identified. The right image
```

Pred: Hyundai Elantra Sedan 2007
Ground Truth: Hyundai Elantra Sedan 2007

Pred: AM General Hummer SUV 2000
Ground Truth: AM General Hummer SUV 2000

Pred: Toyota Sequoia SUV 2012
Ground Truth: Toyota Sequoia SUV 2012

Conclusion	This experiment demonstrates how agentic ai can be used for specific tasks