Project Documentation: AI Image Captioning Application: Enhance Your Images with AI Using Gemini Vision Pro

Version: 1.0.0

Date: June 19, 2025

Author(s): Meet Patel (Topics: 9,10,11,12), Manasvi Khare(Topics: 1,2,3,4), Hammad Khan(Topics: 5,6,7,8)

Contact: meet.23bce10256@vitbhopal.ac.in, manasvi.23bce10232@vitbhopal.ac.in, mohd.23bce10210@vitbhopal.ac.in

1. Overview

In the current digital era, visual content is paramount, particularly on social media platforms. For social media managers, content creators, and businesses, crafting captivating and relevant captions for images is essential yet often labor-intensive. The "AI Image Captioning Application" harnesses the capabilities of Google's Gemini Vision Pro model to automate this task, delivering immediate and appropriate captions for uploaded images. This tool is designed to optimize content creation processes, enabling users to concentrate more on strategy and less on repetitive caption formulation.

2. Challenge Identification & Use Case

- **Challenge:** Manual captioning of images is a monotonous and time-consuming endeavor, especially for individuals or organizations inundated with a high volume of images, such as social media managers, content creators, or e-commerce enterprises. This manual approach can result in inconsistencies in caption quality, delays in content release, and a considerable drain on resources.

3. Use Case: Image Captioning

Consider yourself a social media manager for a well-known brand. Daily, you are tasked with

creating engaging posts that feature striking images and persuasive captions. Developing unique captions for each image can be a lengthy process, particularly when overseeing multiple social media accounts. You require a solution that enables you to swiftly generate appropriate captions for the images you wish to share. This AI Image Captioning Application directly fulfills this requirement by offering an efficient and automated method for generating relevant captions, allowing the social media manager to prioritize strategy and engagement over repetitive caption creation.

4. Generative AI Model & Architecture

The foundation of this AI Image Captioning Application is the Gemini Pro pre-trained model, a cutting-edge Generative AI model from Google. The architecture adheres to a client-server model:

- **User Interface (UI):** Created with Streamlit, providing a user-friendly web interface for image uploads and caption generation initiation.
- **Backend:** Manages communication with the Gemini Pro model. User inputs (image and optional text prompt) are gathered from the UI and sent to the backend using the Google API Key.
- **Gemini Pro Pre-trained Model:** Accepts input via an API call, processes the image and related prompt, and produces a fitting textual caption. The utilized model is gemini-1.5-flash.
- **Output:** The generated caption is sent from the backend to the frontend for formatting and presentation to the user.

5. Data Handling

In this project, the primary "data" being managed is the image uploaded by the user.

- **Input Data:** Images uploaded by users through the Streamlit UI (supported formats: JPG, JPEG, PNG). These images are temporarily processed in memory as byte data before being sent to the Gemini API.

- **Output Data:** The textual captions generated by the Gemini Pro model. There is no long-term storage of user-uploaded images or generated captions within the application. The application processes data on demand, focusing on real-time caption generation.

6. Model Utilization and Testing

Since this project utilizes a pre-trained model (Gemini Pro), no specific training or testing phase is required from the user for the model itself. The Gemini Pro model has been thoroughly pre-trained by Google on an extensive dataset.

- **Testing Focus:**
    - **Prompt Engineering:** Experimenting with various input prompt variations to refine the quality and style of the generated captions. The current input prompt is: "You are an expert in image captioning, so for any image you receive, provide a suitable caption in one phrase." Further adjustments to this prompt could explore different lengths, tones, or specific caption requirements.
    - **Error Management:** Comprehensive testing of the implemented error handling for various API exceptions (e.g., InvalidArgument, FailedPrecondition, ResourceExhausted, DeadlineExceeded) to ensure a reliable user experience.

7. Assessment and Metrics

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

- **Qualitative Evaluation:** The main evaluation metric will be human assessment of the relevance, accuracy, conciseness, and appropriateness of the generated captions for the given image and use case.
- **User Input:** Collecting feedback from target users (e.g., social media managers) regarding the utility and effectiveness of the generated captions in their workflows.
- **Error Rate Tracking:** Monitoring the frequency of API errors encountered to identify potential issues with API key limits, network connectivity, or model availability.

8. Ethical Considerations & Responsible AI

Creating an AI image captioning application requires careful consideration of ethical implications and responsible AI practices:

- **Bias in Captions:** Pre-trained models can inherit biases from their training data. It's essential to recognize that generated captions may sometimes reflect societal biases or stereotypes. Regular monitoring and possibly implementing content moderation or user reporting mechanisms could help mitigate this.
- **Misinterpretation/Misinformation:** While the aim is to provide accurate captions, the model may occasionally misinterpret an image, resulting in inaccurate or misleading captions. It's crucial to clarify that AI-generated captions are suggestions and should be reviewed by a human.
- **Privacy:** The application should not store or misuse user-uploaded images. It should be emphasized that images are processed temporarily for caption generation.
- **Transparency:** Users should be informed that the captions are generated by AI.
- **Harmful Content:** Implement measures to prevent the generation of captions for, or descriptions of, harmful, offensive, or inappropriate content. While the Gemini API likely has built-in safety filters, an additional layer of content filtering may be considered for sensitive applications.

9. Future Developments & Improvements

- **Caption Style Customization:** Allow users to specify desired caption styles (e.g., humorous, professional, poetic, marketing-focused).
- **Multi-language Support:** Expand caption generation capabilities 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 Enhancement:** Implement a mechanism for users to provide feedback on caption quality, which could be used to fine-tune future model iterations or improve prompt engineering.
- **Batch Processing:** Enable users to upload multiple images simultaneously 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:** GitHub Repository
- **Branching Strategy:** A straightforward 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. Getting Started / Usage Instructions

This section outlines how to set up and run the AI Image Captioning Application 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 essential 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:

- streamlit
- google-generativeai
- python-dotenv
- Pillow
- google-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 Instructions:

Follow these steps to install the necessary libraries:

- Open your terminal or Anaconda Prompt.
- Navigate to your project directory using the `cd` command (e.g., `cd path/to/your/project_folder`).
- Install the required libraries by running the following command: `pip install -r requirements.txt`.

11.4 Google API Key Initialization:

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

- **Generate a Google API Key:**
    - Visit the Google AI Studio documentation: Google AI Studio.
    - Sign in with your Google account.
    - Navigate to 'Get an API Key'.
    - Click 'Create API Key' and select 'Generative Language Client' as the project. Choose 'Create API key in existing project'.
    - Copy the newly generated API key.
- **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, managing the UI and interaction with the Gemini model.

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:

- Open your terminal or Anaconda Prompt.
- Navigate to your project folder.
- Run the Streamlit application using the command: `streamlit run app.py`.

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.

CODE:

```
import streamlit as st
import google.generativeai as genai
from PIL import Image
import os
from dotenv import load_dotenv
import io
import PyPDF2
```

```
from streamlit_extras.add_vertical_space import add_vertical_space
# Load environment variables
load_dotenv()
# Configure the page
st.set_page_config(
    page_title="IMAGE TO CAPTION",
    page_icon="📸",
    layout="wide",
    initial_sidebar_state="expanded"
)
# Initialize Google API
def initialize_gemini():
    """Initialize Gemini Pro Vision model with API key"""
    try:
        # Get API key from environment or Streamlit secrets
        api_key = os.getenv("GOOGLE_API_KEY") or st.secrets.get("GOOGLE_API_KEY")

        if not api_key:
            st.error("⚠️ Google API Key not found! Please add it to your .env file or Streamlit
secrets.")
            st.info("Get your API key from: https://makersuite.google.com/app/apikey")
            return None

        genai.configure(api_key=api_key)
        model = genai.GenerativeModel('gemini-1.5-flash')
        return model
```

```
    except Exception as e:
        st.error(f"Error initializing Gemini: {str(e)}")
        return None
def get_gemini_response(model, input_text, image_data):
    """Get response from Gemini Pro Vision model"""
    try:
        if input_text:
            response = model.generate_content([input_text, image_data])
        else:
            response = model.generate_content(image_data)
        return response.text
    except Exception as e:
        st.error(f"Error getting Gemini response: {str(e)}")
        return None
def read_pdf_content(pdf_file):
    """Extract text content from PDF file"""
    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text()
        return text
    except Exception as e:
        st.error(f"Error reading PDF: {str(e)}")
        return None
def prepare_image_data(uploaded_file):
```

```
    """Prepare image data for Gemini Vision"""
    try:
        # Convert uploaded file to bytes
        bytes_data = uploaded_file.getvalue()

        # Create image parts for Gemini
        image_parts = [
            {
                "mime_type": uploaded_file.type,
                "data": bytes_data
            }
        ]
        return image_parts[0]
    except Exception as e:
        st.error(f"Error preparing image data: {str(e)}")
        return None
# Main application
def main():
    # Header
    st.title("📸 IMAGE TO CAPTION")
    st.markdown("### Describe Your Images With AI Using Gemini Vision Pro")

    # Sidebar for API key setup
    with st.sidebar:
        st.header("🔑 Setup")
```

```
# API Key input
api_key_input = st.text_input(
    "Enter your Google API Key:",
    type="password",
    help="Get your API key from Google AI Studio"
)

if api_key_input:
    os.environ["GOOGLE_API_KEY"] = api_key_input
    st.success("✅ API Key set successfully!")

st.markdown("---")
st.markdown("### 📋 Instructions")
st.markdown("""
1. Enter your Google API Key above
2. Upload an image (JPG, JPEG, PNG)
3. Optionally add a custom prompt
4. Click 'Generate Caption' to get AI description
""")

st.markdown("---")
st.markdown("### 🎯 Use Cases")
st.markdown("""
- **Social Media**: Generate engaging captions
- **Content Creation**: Describe images for blogs
- **Accessibility**: Create alt text for images
```

```
    - **E-commerce**: Product descriptions
    """)

# Initialize Gemini model
model = initialize_gemini()

if model is None:
    st.warning("Please configure your Google API Key to continue.")
    return

# Main content area
col1, col2 = st.columns([1, 1])

with col1:
    st.header("📤 Upload & Configure")

    # Custom prompt input
    input_prompt = st.text_area(
        "Custom Prompt (Optional):",
        placeholder="e.g., 'Describe this image for social media', 'Create a creative caption', 'What's happening in this image?'",
        height=100
    )

    # File uploader
    uploaded_file = st.file_uploader(
```

```
    "Choose an image file",
    type=['jpg', 'jpeg', 'png'],
    help="Upload an image to generate captions"
)

# Quick prompt buttons
st.markdown("#### 🚀 Quick Prompts")
col_btn1, col_btn2 = st.columns(2)

with col_btn1:
    if st.button("📱 Social Media Caption"):
        input_prompt = "Create an engaging social media caption for this image"

with col_btn2:
    if st.button("🎨 Creative Description"):
        input_prompt = "Provide a creative and detailed description of this image"

col_btn3, col_btn4 = st.columns(2)

with col_btn3:
    if st.button("🛍️ Product Description"):
        input_prompt = "Create a product description based on this image"

with col_btn4:
    if st.button("♿ Alt Text"):
        input_prompt = "Generate accessibility alt text for this image"
```

```
with col2:
    st.header("🖼️ Image Preview")

    if uploaded_file is not None:
        # Display the uploaded image
        image = Image.open(uploaded_file)
        st.image(image, caption="Uploaded Image", use_column_width=True)

        # Image details
        st.markdown("**Image Details:**")
        st.write(f"- **Filename:** {uploaded_file.name}")
        st.write(f"- **File size:** {uploaded_file.size} bytes")
        st.write(f"- **Dimensions:** {image.size[0]} x {image.size[1]} pixels")
        st.write(f"- **Format:** {image.format}")
    else:
        st.info("👆 Please upload an image to see preview")

# Generate caption section
if uploaded_file is not None:
    add_vertical_space(2)

    st.header("🎯 Generate Caption")

    col_gen1, col_gen2, col_gen3 = st.columns([1, 1, 1])
```

```
with col_gen1:
    generate_button = st.button(
        "🚀 Generate Caption",
        type="primary",
        use_container_width=True
    )

with col_gen2:
    one_phrase_button = st.button(
        "📝 Give me caption in one phrase",
        use_container_width=True
    )

with col_gen3:
    detailed_button = st.button(
        "📚 Detailed Analysis",
        use_container_width=True
    )

# Process the image and generate caption
if generate_button or one_phrase_button or detailed_button:
    with st.spinner("🤖 AI is analyzing your image..."):
        try:
            # Prepare image data
            image_data = prepare_image_data(uploaded_file)
```

```
            if image_data:
                # Set prompt based on button clicked
                if one_phrase_button:
                    prompt = "Give me the caption in one phrase"
                elif detailed_button:
                    prompt = "Provide a detailed analysis and description of this image,
including objects, people, setting, mood, and any notable details"
                else:
                    prompt = input_prompt if input_prompt else "Describe this image in
detail"

                # Get response from Gemini
                response = get_gemini_response(model, prompt, image_data)

                if response:
                    st.success("✅ Caption generated successfully!")

                    # Display results
                    st.header("📋 Generated Caption")

                    # Create a nice formatted output
                    st.markdown(f"""
                    <div style="
                        background-color: #f0f2f6;
                        padding: 20px;
                        border-radius: 10px;
```

```
                    border-left: 5px solid #1f77b4;
                    margin: 10px 0;
                ">
                    <h4 style="margin-top: 0; color: #1f77b4;">🎯 AI Generated
Caption:</h4>
                    <p style="font-size: 16px; line-height: 1.6; margin-bottom:
0;">{response}</p>
                </div>
                """, unsafe_allow_html=True)

                # Copy to clipboard functionality
                st.text_area(
                    "Copy this caption:",
                    value=response,
                    height=100,
                    help="Select all text and copy (Ctrl+A, Ctrl+C)"
                )

            else:
                st.error("Failed to generate caption. Please try again.")

        except Exception as e:
            st.error(f"An error occurred: {str(e)}")

    # Footer
    add_vertical_space(3)
```

```
    st.markdown("---")
    st.markdown("""
    <div style="text-align: center; color: #666;">
        <p>🤖 Powered by Google Gemini Vision Pro | Built with Streamlit</p>
        <p>Perfect for social media managers, content creators, and accessibility
needs!</p>
    </div>
    """, unsafe_allow_html=True)
# Additional utility functions
def create_env_file():
    """Create a sample .env file"""
    env_content = """# Google API Configuration
GOOGLE_API_KEY=your_google_api_key_here
# App Configuration
APP_NAME=AI Image Captioning App
DEBUG=False
"""

    with open('.env.example', 'w') as f:
        f.write(env_content)
def setup_instructions():
    """Display setup instructions"""
    st.markdown("""
    ## 🚀 Setup Instructions

    ### 1. Install Dependencies
```

```bash
pip install -r requirements.txt
```

2. Get Google API Key

1. Go to [Google AI Studio](https://makersuite.google.com/app/apikey)
2. Create a new API key
3. Copy the API key

3. Configure Environment

1. Create a `.env` file in your project directory
2. Add your API key:
```
GOOGLE_API_KEY=your_api_key_here
```

4. Run the Application

```bash
streamlit run main.py
```

5. Deploy to Streamlit Cloud

1. Push your code to GitHub
2. Connect to [Streamlit Cloud](https://streamlit.io/cloud)
3. Add your API key to Streamlit secrets
""")

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

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: Google Gemini API
- Streamlit Documentation: Streamlit Docs

12.2 Libraries Used:

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