**VISUAL LANGUAGE MODEL**

**1. Introduction**

This project aims to generate images based on user-provided text prompts using Stable Diffusion and then analyze the generated images using OpenAI's CLIP model. The system compares the generated image against predefined object categories to determine the best matching description.

**2. Objectives**

* Generate images based on user-provided text descriptions.
* Use the CLIP model to analyze the generated images.
* Identify the most relevant object category for the generated image.

**3. Technologies Used**

* **Python**: Programming language for implementation.
* **Torch**: Deep learning framework used for CLIP.
* **CLIP** **(Contrastive Language-Image Pretraining):** A model for associating images and text descriptions.
* **Stable** **Diffusion**: A text-to-image generation model.
* **PIL (Python Imaging Library):** For image processing.
* **NumPy**: For numerical computations.
* **Google** **Colab**: Execution environment.

**4. System Architecture**

The system follows these steps:

1. Load the CLIP model and Stable Diffusion.
2. Accept a text prompt from the user.
3. Generate an image using Stable Diffusion based on the input prompt.
4. Preprocess the generated image for CLIP analysis.
5. Compare the generated image with predefined text categories using CLIP.
6. Output the best-matching object category.

**5. Implementation Details**

The implementation consists of:

* Loading the CLIP and Stable Diffusion models.
* Processing user input for image generation.
* Using CLIP to compute similarity between the generated image and predefined categories.
* Identifying and displaying the most relevant match.

**5.1 Code Implementation**

**Step 1: Install Dependencies and Import Libraries**

!pip install torch torchvision

!pip install git+https://github.com/openai/CLIP.git

import torch

import clip

import os

import random

import pandas as pd

from PIL import Image

from IPython.display import display

from google.colab import files

**Explanation**: This section installs the required dependencies and imports necessary libraries. torch and clip are used for model operations, PIL for image handling, and pandas for working with labels.

**Step 2: Load the CLIP Model**

# Load CLIP model

device = "cuda" if torch.cuda.is\_available() else "cpu"

model, preprocess = clip.load("ViT-B/32", device=device)

**Explanation:** This loads the CLIP model and selects the appropriate device (GPU if available, otherwise CPU) for computation.

**Step 3: Load Labels from CSV**

# Load labels from CSV

csv\_path = "/content/labels.csv"

df = pd.read\_csv(csv\_path)

labels = df['Label'].tolist()

**Explanation:** This reads object category labels from a CSV file, which will be used for classification.

**Step 4: Upload and Process Images**

# Upload images to Colab

print("Please upload images (JPG, JPEG, PNG, WEBP, GIF, BMP) for classification...")

uploaded\_files = files.upload()

# Get list of supported image files

supported\_formats = (".jpg", ".jpeg", ".png", ".webp", ".gif", ".bmp")

image\_files = [file for file in uploaded\_files.keys() if file.lower().endswith(supported\_formats)]

# Ensure images are available

if not image\_files:

raise ValueError("No valid image files were uploaded. Please upload at least one.")

**Explanation:** This section prompts the user to upload images and verifies that at least one valid image file is available.

**Step 5: Select and Preprocess an Image**

# Select a random image

random\_image\_path = random.choice(image\_files)

print(f"\nSelected Random Image: {random\_image\_path}")

image = Image.open(random\_image\_path).convert("RGB")

image\_processed = preprocess(image).unsqueeze(0).to(device)

**Explanation:** This selects a random image from the uploaded files, converts it to RGB format, and preprocesses it for CLIP analysis.

**Step 6: Display the Selected Image**

# Display the selected image

image.show()

text\_inputs = clip.tokenize(labels).to(device)

print(f"\nDisplaying Selected Image: {random\_image\_path}")

display(image)

**Explanation:** This section displays the selected image and tokenizes the text labels for comparison.

**Step 7: Perform Zero-Shot Classification**

# Perform zero-shot classification

with torch.no\_grad():

image\_features = model.encode\_image(image\_processed)

text\_features = model.encode\_text(text\_inputs)

similarity = torch.cosine\_similarity(image\_features, text\_features, dim=-1)

**Explanation:** This step extracts feature embeddings from both the image and text labels and computes their similarity scores using cosine similarity.

**Step 8: Identify and Display the Best Match**

# Get the best match

best\_match = labels[similarity.argmax().item()]

print(f"\nPredicted Label: {best\_match}")

# Display similarity scores for all labels

print("\nSimilarity Scores:")

for label, score in zip(labels, similarity.tolist()):

print(f"{label}: {score:.4f}")

**Explanation:** This section identifies the label with the highest similarity score as the best match and displays similarity scores for all labels.

**6. Applications**

This system has several practical applications, including:

* **Content-based Image Retrieval (CBIR):** Finding similar images based on text descriptions.
* **Automated Image Tagging:** Assigning labels to images for efficient cataloging.
* **AI-Powered Search Engines:** Enhancing search functionalities with text-image understanding.
* **Medical Imaging:** Categorizing and analyzing medical scans.
* Creative Design & Media: Generating and classifying images for design applications.

**7. Results**

The system successfully:

* Accepts user input for text descriptions.
* Generates high-quality images using Stable Diffusion.
* Identifies the most relevant category using CLIP.

**8. Conclusion**

This project demonstrates the potential of generative AI models for creating and analyzing images. By combining Stable Diffusion and CLIP, we can generate images and categorize them with reasonable accuracy. Future improvements may include fine-tuning the models, expanding the category list, and integrating real-world datasets for better generalization.

**9. Future Enhancements**

* Improving text prompt engineering for better image generation.
* Fine-tuning CLIP to improve image-text matching.
* Adding more diverse categories for classification.
* Implementing a GUI for better user interaction.

**10. Project Partner Contributions**

**Partner 1: Prahalad Pal**

* Implemented and optimized the image generation process using Stable Diffusion.
* Integrated user input handling and preprocessing for the CLIP model.
* Conducted testing and debugging to improve model performance.

**Partner 2: Arun Maity**

* Developed the classification logic using CLIP for image-text similarity computation.
* Designed and structured the system architecture and dataset management.
* Documented the project, prepared reports, and suggested future improvements.

**11. References**

* **OpenAI's CLIP:** [**https://openai.com/research/clip**](https://openai.com/research/clip)
* **PyTorch:** [**https://pytorch.org/**](https://pytorch.org/)