Model Documentation

# Overview

This Python script leverages two transformer models for image captioning and zero-shot classification:  
1. \*\*BLIP (Bootstrapped Language-Image Pre-training)\*\*: Used for generating captions for images.  
2. \*\*CLIP (Contrastive Language–Image Pretraining)\*\*: Used for zero-shot classification, matching images to categories.  
  
The script also demonstrates an ensemble method that combines predictions from both models and an external API to determine the final classification of an image. The external API is a placeholder that simulates additional content moderation.

# Dependencies

The script requires the following Python libraries:  
- `requests`: For fetching images from URLs.  
- `Pillow (PIL)`: For processing and handling images.  
- `transformers`: For using pre-trained BLIP and CLIP models.  
- `torch`: For tensor operations and model inference.  
  
To install these dependencies:  
```bash  
pip install requests Pillow transformers torch  
```

# Model Initialization

## 1. BLIP for Image Captioning

The BLIP model is used to generate detailed captions for input images:  
```python  
blip\_processor = BlipProcessor.from\_pretrained('Salesforce/blip-image-captioning-base')  
blip\_model = BlipForConditionalGeneration.from\_pretrained('Salesforce/blip-image-captioning-base')  
```

## 2. CLIP for Zero-Shot Classification

The CLIP model matches images to text labels for classification:  
```python  
clip\_model = CLIPModel.from\_pretrained('openai/clip-vit-base-patch32')  
clip\_processor = CLIPProcessor.from\_pretrained('openai/clip-vit-base-patch32')  
```

## 3. Categories for Classification

A set of predefined categories is used for classification using the CLIP model:  
```python  
categories = [  
 'safe', 'nsfw', 'abusive', 'erotic', 'violent',   
 'hate speech', 'graphic content', 'explicit nudity',   
 'suggestive content', 'bullying', 'harassment',   
 'terrorism', 'extremism', 'drug use', 'self-harm',   
 'spam', 'misinformation', 'trolling', 'racist content',   
 'child exploitation', 'animal abuse', 'violent extremism',   
 'cultural insensitivity', 'profanity', 'graphic injury',   
 'illegal activities'  
]  
```

# Functions

## 1. get\_image\_caption\_blip(image)

Generates a caption for the input image using the BLIP model:  
- \*\*Parameters\*\*: `image`: A PIL Image object.  
- \*\*Returns\*\*: A string containing the generated caption.  
```python  
def get\_image\_caption\_blip(image):  
 inputs = blip\_processor(images=image, return\_tensors='pt')  
 with torch.no\_grad():  
 output = blip\_model.generate(\*\*inputs)  
 caption = blip\_processor.decode(output[0], skip\_special\_tokens=True)  
 return caption  
```

## 2. classify\_image\_clip(image, categories)

Classifies the input image into one of the predefined categories using the CLIP model:  
- \*\*Parameters\*\*:  
 - `image`: A PIL Image object.  
 - `categories`: A list of classification labels.  
- \*\*Returns\*\*: The category with the highest predicted probability.  
```python  
def classify\_image\_clip(image, categories):  
 inputs = clip\_processor(text=categories, images=image, return\_tensors='pt', padding=True)  
 outputs = clip\_model(\*\*inputs)  
 logits\_per\_image = outputs.logits\_per\_image  
 probs = logits\_per\_image.softmax(dim=1)  
 predicted\_idx = probs.argmax().item()  
 return categories[predicted\_idx]  
```

## 3. get\_external\_api\_classification(image\_url)

Placeholder function to simulate external API calls for content moderation. It takes an image URL as input and returns a simulated classification category:  
- \*\*Parameters\*\*: `image\_url`: A string representing the image URL.  
- \*\*Returns\*\*: A string containing the predicted category from the API.  
```python  
def get\_external\_api\_classification(image\_url):  
 api\_response = {'predicted\_category': 'graphic content'}  
 return api\_response['predicted\_category']  
```

## 4. ensemble\_predict(image, image\_url)

This function combines predictions from the BLIP model, the CLIP model, and an external API, and returns the image caption and final predicted category:  
- \*\*Parameters\*\*:  
 - `image`: A PIL Image object.  
 - `image\_url`: The URL of the image.  
- \*\*Returns\*\*: The image caption and the final predicted category.  
```python  
def ensemble\_predict(image, image\_url):  
 predictions = []  
 caption = get\_image\_caption\_blip(image)  
 predictions.append(caption)  
 clip\_prediction = classify\_image\_clip(image, categories)  
 predictions.append(clip\_prediction)  
 api\_prediction = get\_external\_api\_classification(image\_url)  
 predictions.append(api\_prediction)  
 final\_prediction = max(set(predictions), key=predictions.count)  
 return caption, final\_prediction  
```

# Example Usage

To run the ensemble prediction on an image from a URL:  
1. Replace the `url` variable with your desired image URL.  
2. Fetch the image using the `requests` library.  
3. Run the `ensemble\_predict` function to get the caption and predicted category.  
```python  
url = 'https://example.com/image.jpg'  
response = requests.get(url, stream=True)  
if response.status\_code == 200 and 'image' in response.headers['Content-Type']:  
 image = Image.open(BytesIO(response.content))  
 caption, predicted\_category = ensemble\_predict(image, url)  
 print(f'Image Caption: {caption}')  
 print(f'Predicted Category: {predicted\_category}')  
else:  
 print('Failed to retrieve a valid image.')  
```

# Conclusion

This script demonstrates the use of transformer-based models (BLIP and CLIP) for image captioning and classification. The ensemble approach combines multiple predictions to improve content understanding and moderation.