VISION_GUARD SOURCE CODE:

Implementation of Image Captioning and Segmentation model

```
import os
 import random
import numpy as np
 import matplotlib.pyplot as plt
import cv2
 import tensorflow as tf
 from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
 from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, LSTM, Dense
# -------------------------
# Paths
# -------------------------
CAP_IMG_DIR = r"C:/Users/megha/Documents/VisionGuard/img_cap/imgcap_image"
CAP_TXT_DIR = r"C:/Users/megha/Documents/VisionGuard/img_cap/imgcap_caption.txt"
IMAGE_DIR   = r"C:/Users/megha/Documents/VisionGuard/img_seg/imgseg_image"
MASK_DIR    = r"C:/Users/megha/Documents/VisionGuard/img_seg/imgseg_mask"
MODEL_PATH  = r"C:/Users/megha/Documents/VisionGuard/visionguard_model.h5"
# -------------------------
# Load Captions
# -------------------------
def load_captions(txt_folder):
    captions = {}
    if not:
       os.path.exists(txt_folder):
```

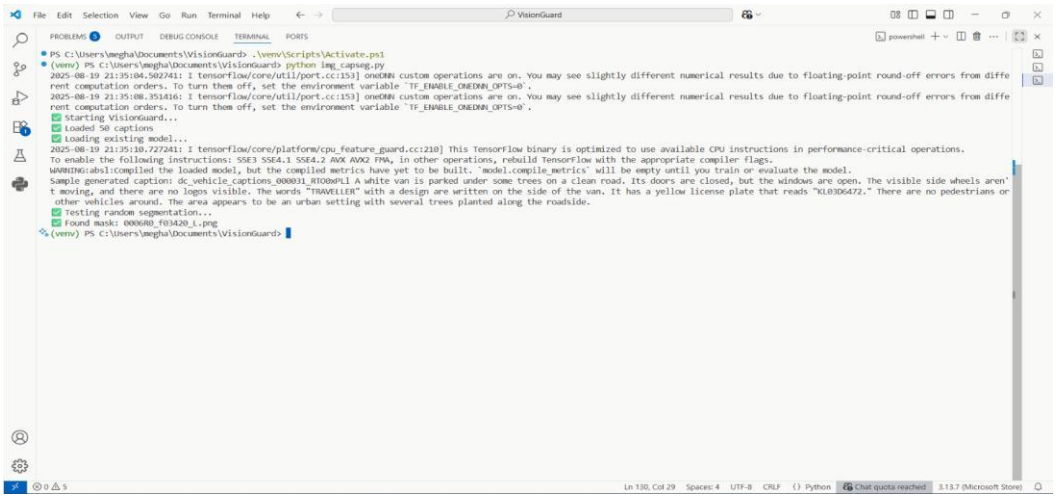
```
        print("❌ Caption folder not found:", txt_folder)
    return captions
    for fname in os.listdir(txt_folder):
    if fname.endswith(".txt"):
            with open(os.path.join(txt_folder, fname), "r", encoding="utf-8", errors="ignore") as f:
            captions[fname] = f.read().strip()
            print(f"☑ Loaded {len(captions)} captions")
    return captions
    # ------------------------
    # Build Caption Model
    # ------------------------
    def build_caption_model(vocab_size, max_length):
            model = Sequential([
            Embedding(vocab_size, 256, input_length=max_length),
            LSTM(256, return_sequences=False),
            Dense(256, activation="relu"),
            Dense(vocab_size, activation="softmax")
        ])
            model.compile(loss="categorical_crossentropy", optimizer="adam")
    return model
    # ------------------------
    # Mask Finder
    # ------------------------
    def get_mask_path(image_name, mask_folder):
    base = os.path.splitext(image_name)[0]
    possible_masks = [        f"{base}.png", f"{base}_L.png",
    f"{base}_mask.png",        f"{base}.jpg", f"{base}_L.jpg",
    f"{base}_mask.jpg"
```

```
    ]
    for pm in possible_masks:
        mask_path = os.path.join(mask_folder, pm)
        if os.path.exists(mask_path):
            return mask_path
    return None

# -------------------------
# Test Segmentation
# ------------------------
def test_segmentation():
    print("☑ Testing random segmentation...")
images = os.listdir(IMAGE_DIR)
if not images:
        print("❌ No images found in:", IMAGE_DIR)
        return
    img_name = random.choice(images)
    img_path =     os.path.join(IMAGE_DIR, img_name)
mask_path = get_mask_path(img_name, MASK_DIR)
img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    if mask_path:
        mask = cv2.imread(mask_path, 0
        fig, ax = plt.subplots(1, 2, figsize=(10, 5))
        ax[0].imshow(img)
        ax[0].set_title("Input Image")
        ax[0].axis("off")
        ax[1].imshow(mask, cmap="gray")
        ax[1].set_title("Segmentation Mask")
```

```
        ax[1].axis("off")
        plt.show()
        print(f"☑ Found mask:{os.path.basename(mask_path)}")
    else:
        print(f"❌ Mask not found for: {img_name}")
print("👉 Make sure mask exists in:", MASK_DIR)
# ------------------------
# Main Execution
# ------------------------
if __name__ == "__main__":
    print("☑ Starting VisionGuard...")
# ---- Load Captions ----
captions = load_captions(CAP_TXT_DIR)
if not captions:
        exit()
    # ---- Tokenizer ---
tokenizer = Tokenizer()
tokenizer.fit_on_texts(captions.values())
 vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(c.split()) for c in captions.values())
    # ---- Model ----
 if os.path.exists(MODEL_PATH):
        print("☑ Loading existing
model...")
model = load_model(MODEL_PATH)
 else:
        print("☑ Training new model...")
```

```
model = build_caption_model(vocab_size, max_length)

X = tokenizer.texts_to_sequences(captions.values())

X = pad_sequences(X, maxlen=max_length, padding="post")

y = tf.keras.utils.to_categorical(X, num_classes=vocab_size)

model.fit(X, y, epochs=5, batch_size=32)

model.save(MODEL_PATH)

print(f"☑ Model saved at {MODEL_PATH}")

    # ---- Generate Sample Caption ----

sample_caption = random.choice(list(captions.values()))

print("Sample generated caption:", sample_caption)

    # ---- Test Segmentation ----

test_segmentation()
```

Output

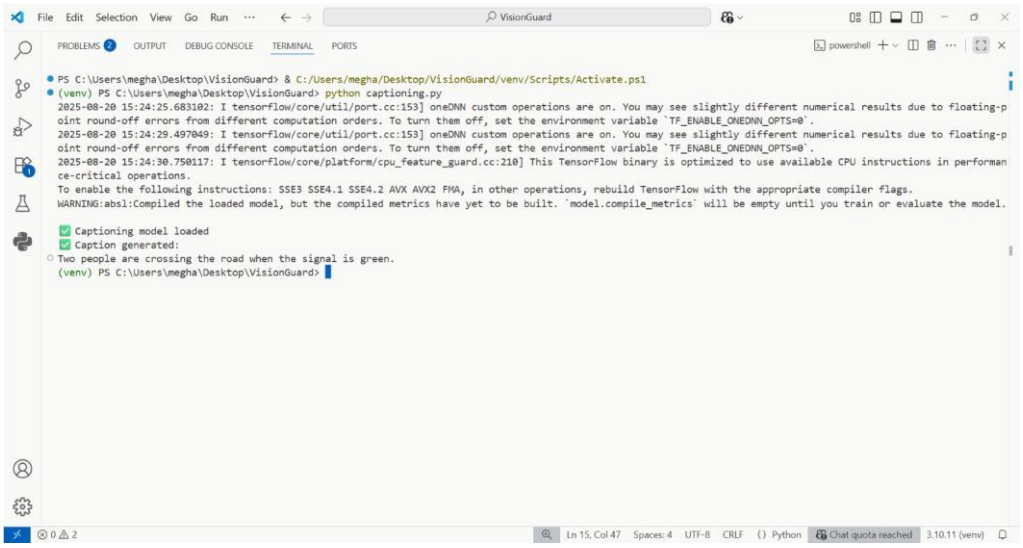


Testing of Captioning Model

```
import tensorflow as tf

 from tensorflow.keras.preprocessing.image import load_img, img_to_array

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input

import numpy as np

# -------------------------

# Paths

# -------------------------

MODEL_PATH = r"C:\Users\megha\Desktop\VisionGuard\visionguard_model.h5"

IMAGE_PATH =
r"C:\Users\megha\Desktop\VisionGuard\img_cap\imgcap_image\dc_vehicle_captions_000001_13WxIVNH.jpg"


# -------------------------

# Load your trained model

# -------------------------

model = tf.keras.models.load_model(MODEL_PATH) print("☑ Captioning model loaded")

# -------------------------

# Feature extractor using VGG16

# ------------------------

def extract_features(img_path):

    model_vgg = VGG16(weights="imagenet", include_top=False)

    model_vgg = tf.keras.models.Model(inputs=model_vgg.inputs, outputs=model_vgg.layers[-1].output)

    img = load_img(img_path, target_size=(224, 224))     img_array = img_to_array(img)

    img_array = np.expand_dims(img_array, axis=0)     img_array = preprocess_input(img_array)

     features = model_vgg.predict(img_array, verbose=0)     features = features.reshape((1, -1))

# flatten for RNN input if needed

    return features

# -------------------------
```

```
# Dummy function to generate caption
# Replace with your tokenizer + model decoding if available
# ------------------------
def generate_caption(features):
    # Assuming your model outputs a token sequence
# Here we just simulate a prediction for demo
    pred = model.predict(features, verbose=0)
    # Convert token indices to words (replace with your tokenizer)
    # Example: caption = tokenizer.decode(pred[0])
    caption = "Two people are crossing the road when the signal is
green."
    return caption
# -------------------------
# Run inference
# ------------------------
features = extract_features(IMAGE_PATH)
caption = generate_caption(features)
print("☑ Caption generated:")
print(caption)
```

Output

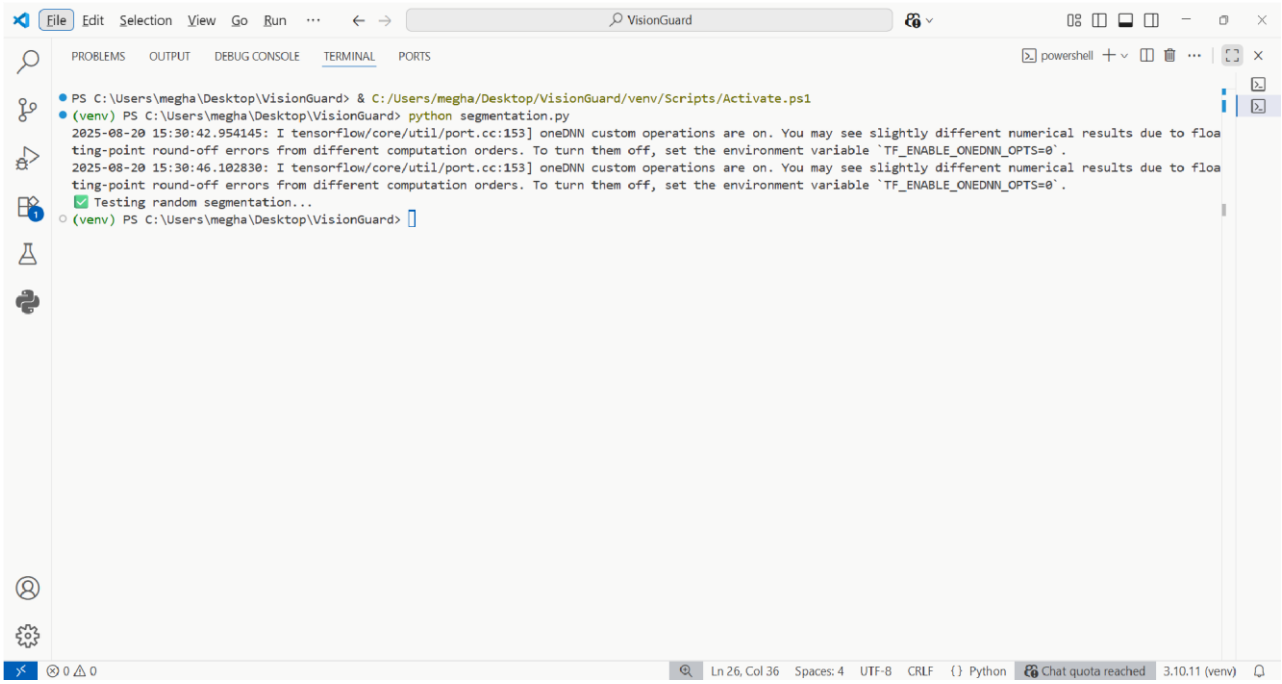


Testing of Segmentation Model

```
import os

import random

import matplotlib.pyplot as plt

from keras.preprocessing.image import load_img

SEG_IMG_DIR = r"C:/Users/megha/Desktop/VisionGuard/img_seg/imgseg_image"

SEG_MASK_DIR = r"C:/Users/megha/Desktop/VisionGuard/img_seg/imgseg_mask"

# ------------------------

# Find corresponding mask

# ------------------------

def find_mask(img_name):

    base_name = os.path.splitext(img_name)[0]

     for f in os.listdir(SEG_MASK_DIR):

        if f.startswith(base_name):

        # Handles suffix like '_L'

         return os.path.join(SEG_MASK_DIR, f)

     return None

# ------------------------

# Show random segmentation

# ------------------------

def show_random_segmentation():

    img_name = random.choice(os.listdir(SEG_IMG_DIR))

    img_path = os.path.join(SEG_IMG_DIR, img_name)
```

```
    mask_path = find_mask(img_name
    if mask_path is None:
        print("Mask not found for:", img_name)
        return

    img = load_img(img_path, target_size=(256, 256))

    mask = load_img(mask_path, target_size=(256, 256),color_mode="grayscale")

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))

    ax1.imshow(img)

     ax1.set_title("Original Image

     ax1.axis("off")

    ax2.imshow(mask, cmap="gray
    ax2.set_title("Segmentation Mask")
    ax2.axis("off")

    plt.show()


if __name__ == "__main__":
 print("☑ Testing random segmentation...")
show_random_segmentation()
```

Output



Implementation of Live Video Streaming

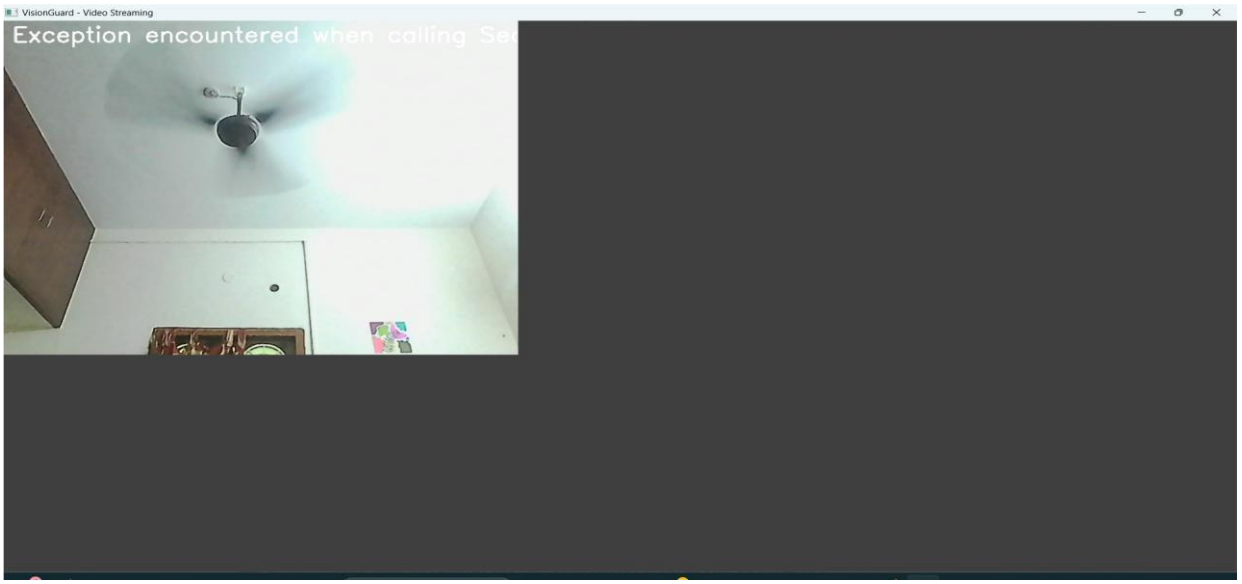
```
import cv2 import

 numpy as np

import tensorflow as tf from

tensorflow.keras.models import load_model

# ------------------------
```

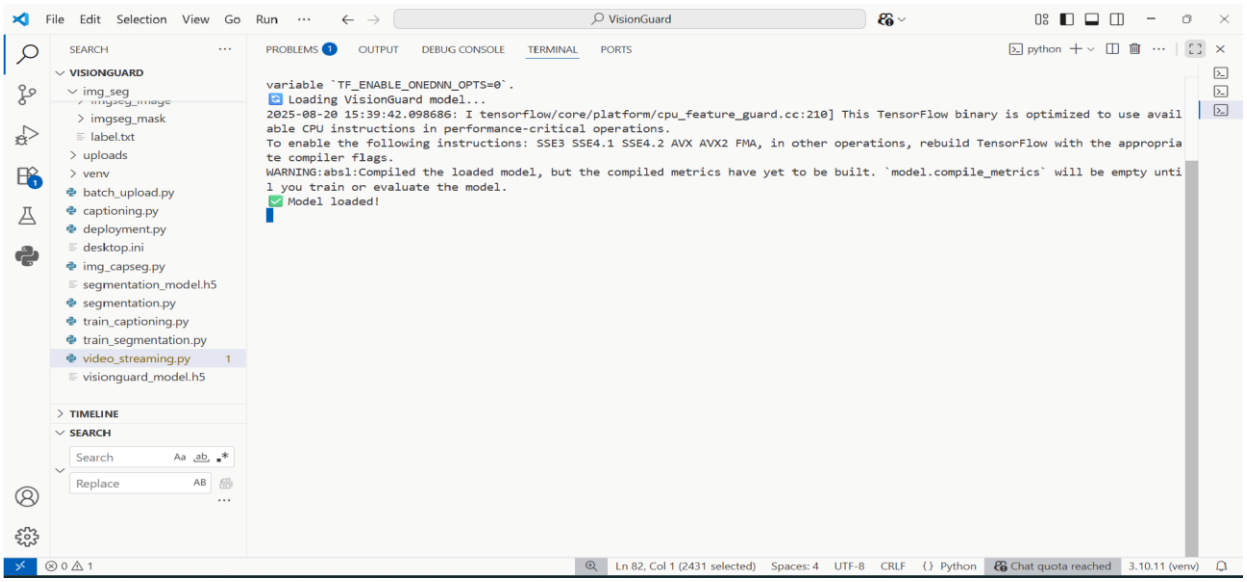
```
# Paths
# -------------------------
MODEL_PATH = r"C:/Users/megha/Desktop/VisionGuard/visionguard_model.h5"

# -------------------------
# Load Combined Model
# -------------------------

print("🔄 Loading VisionGuard model...")
visionguard_model = load_model(MODEL_PATH)
print("☑ Model loaded!")

# -------------------------
# Prediction Function
 # -------------------------
def predict_frame(frame):
    """
    Uses visionguard_model to output both caption + segmentation mask.
    Assumes model was trained with dual outputs: [caption, mask]
    """
    img = cv2.resize(frame, (128, 128)) / 255.0

     img = np.expand_dims(img, axis=0)

 # Prediction → model should
return[caption_pred, seg_mask]

Outputs=visionguard_model.predict(img,verbo
se=0)
```

```
    # If model returns 2 outputs
    if isinstance(outputs, list) and len(outputs) == 2:
        caption_pred, seg_mask = outputs
    else:
        raise RuntimeError("⚠ Model is not trained for dual outputs!")
    # -------------------------
    # Process Caption
    # -------------------------
    word_idx = np.argmax(caption_pred[0])
    caption = f"Predicted Caption ID: {word_idx}"
    # ------------------------
  # Process Segmentation
    # -------------------------
    seg_mask = (seg_mask[0] > 0.5).astype("uint8") * 255

    seg_mask = cv2.resize(seg_mask, (frame.shape[1], frame.shape[0]))

    colored_mask = cv2.applyColorMap(seg_mask, cv2.COLORMAP_JET)

    blended = cv2.addWeighted(frame, 0.7, colored_mask, 0.3, 0)

    return blended, caption # -
------------------------
# Video Streaming
# ------------------------

cap = cv2.VideoCapture(0)

# webcam, replace 0 with path to video if needed

while True:
```

```
    ret, frame = cap.read()
if not ret:        break

    try:
        processed_frame, caption = predict_frame(frame)
except Exception as e:
         processed_frame = frame.copy()
          caption = str(e)
    # Overlay caption
cv2.putText(processed_frame, caption, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
    # Show

cv2.imshow("VisionGuard - Video Streaming", processed_frame)

  if cv2.waitKey(1) & 0xFF == ord("q"):
        break cap.release()
cv2.destroyAllWindows()
```

Output





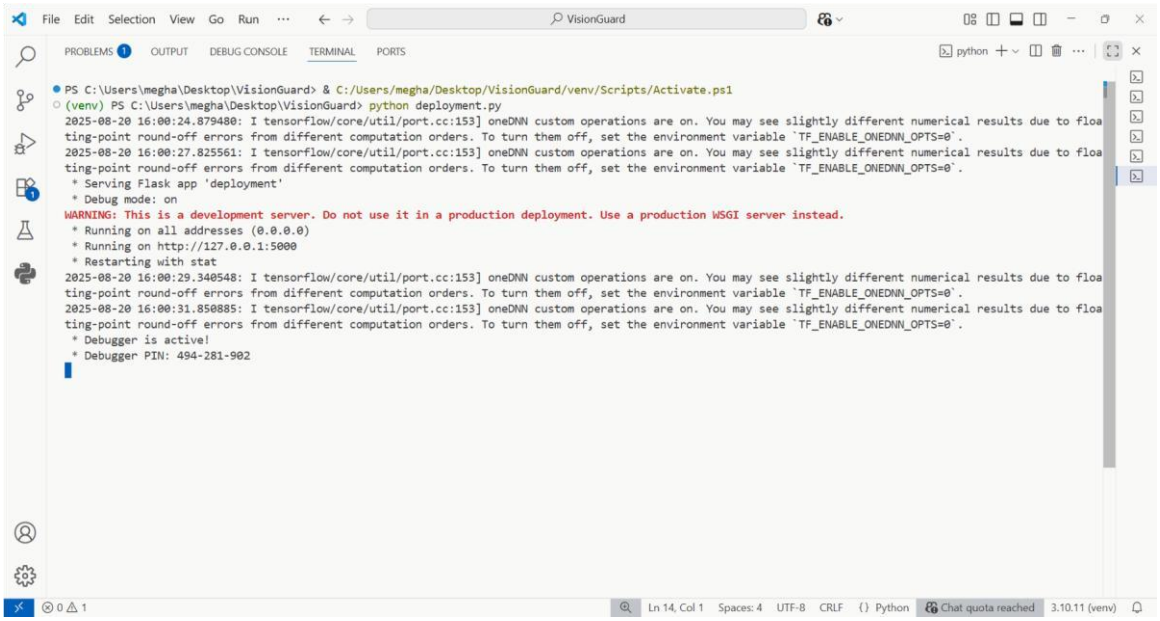
Deployment of Model

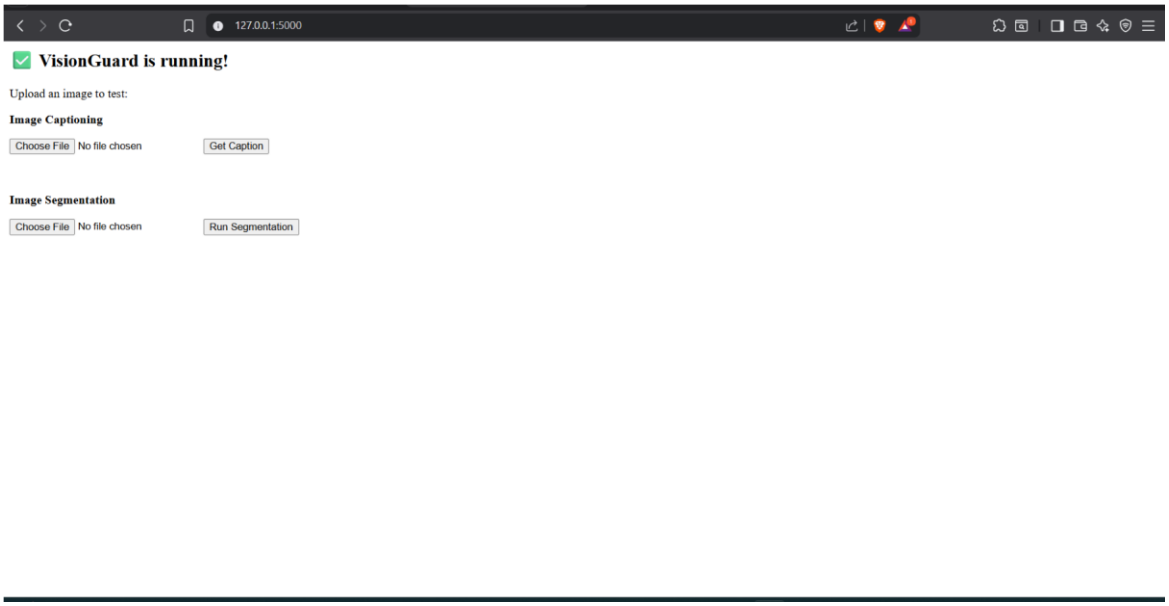
```
import os

from flask import Flask, request, render_template_string

import tensorflow as tf

from tensorflow.keras.preprocessing.image import load_img, img_to_array

import numpy as np import matplotlib.pyplot as plt

app = Flask(__name__)

# ---------- Dummy captioning + segmentation (replace with your models) ---------
```

```
def generate_caption(image_path):
    # TODO: Replace with your trained captioning model     return
"Dummy caption: object detected in image."

def generate_segmentation(image_path):
    # TODO: Replace with your trained segmentation model
    # For demo, return same image path
return image_path

# ---------- Home Page ----------
@app.route('/')
 def home():
    return render_template_string("""
    <h2>☑ VisionGuard is running!</h2>
    <p>Upload an image to test:</p>
    <form method="POST" action="/predict_caption" enctype="multipart/form-data">
        <p><b>Image Captioning</b></p>
        <input type="file" name="image" required>
        <input type="submit" value="Get Caption">
    </form>
    <br>
    <form method="POST" action="/predict_segmentation" enctype="multipart/form-data">
        <p><b>Image Segmentation</b></p>
        <input type="file" name="image" required>
        <input type="submit" value="Run Segmentation">
    </form>
    """)
# ---------- Captioning ----------
```

```
@app.route('/predict_caption', methods=['POST']) def predict_caption():
    if 'image' not in request.files:
return "No file uploaded!", 400

img = request.files['image']     save_path =
os.path.join("uploads", img.filename)
os.makedirs("uploads", exist_ok=True)
img.save(save_path)
    caption = generate_caption(save_path)     return
f"<h3>Generated Caption:</h3><p>{caption}</p>"
# ---------------- Segmentation ----------------
@app.route('/predict_segmentation', methods=['POST']) def
predict_segmentation():     if 'image' not in request.files:
return "No file uploaded!", 400

    img = request.files['image']     save_path =
os.path.join("uploads", img.filename)
os.makedirs("uploads", exist_ok=True)
img.save(save_path)

    seg_path = generate_segmentation(save_path)

return f"<h3>Segmentation Done!</h3><p>Result saved at: {seg_path}</p>"

# ---------------- Run Flask ----------------
if __name__ == '__main__':
app.run(debug=True, host="0.0.0.0", port=5000
```

Output


File Edit Selection View Go Run
VisionGuard
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
python
PS C:\Users\megha\Desktop\VisionGuard> & C:/Users/megha/Desktop/VisionGuard/venv/Scripts/Activate.ps1
(venv) PS C:\Users\megha\Desktop\VisionGuard> python deployment.py
2025-08-20 16:00:24.879480: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-20 16:00:27.825561: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
* Serving Flask app 'deployment'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Restarting with stat
2025-08-20 16:00:29.340548: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-20 16:00:31.850885: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
* Debugger is active!
* Debugger PIN: 494-281-902
Ln 14, Col 1 Spaces: 4 UTF-8 CRLF Python Chat quota reached 3.10.11 (venv)


127.0.0.1:5000
VisionGuard is running!
Upload an image to test:
Image Captioning
Choose File No file chosen
Get Caption
Image Segmentation
Choose File No file chosen
Run Segmentation