NETWORKS FINAL PROJECT

Neil Xu, Suraj Anand
Brown University
neil_xu@brown.edu, suraj_anand@brown.edu

1 Introduction: Project Goals and Achievements

1.1 Goals

Our project's objective was to develop an HTTP API for meme generation using images uploaded by users. We aimed to implement various themed endpoints and achieve high-quality image-caption matching using machine learning techniques.

1.2 Achievements

We successfully created an API capable of generating memes with appropriate captions. The backend integrates advanced machine learning models for image processing and caption generation, exceeding our initial goals.

2 Design/Implementation: System Overview

2.1 Backend Implementation

We used Python, Flask, and CLIP (Contrastive Language-Image Pretraining) model for our backend. The key components include:

Flask Backend Our backend is built using Flask, which handles image uploads, file validations, and interactions with the machine learning model for caption generation.

Endpoints

- `@app.route('/upload', methods=['POST'])` handles image uploads.
- `@app.route('/regenerate')` allows for caption regeneration without re-uploading the image.
- `@app.route('/makeImage')` uses pillow to create a meme with the image and caption.

CLIP Model Integration We utilized the CLIP model from OpenAI, running on either CUDA or CPU, to compute image and text features. This model is central to matching images with relevant captions.

Meme Caption Generation Using a predefined list of humorous one-liners, we precomputed text embeddings to quickly match with uploaded images. This caching mechanism sped up our backend greatly.

```
import torch
import clip
from PIL import Image
from io import BytesIO

device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)

def precompute_text_embeddings(captions, model, device):
```

```
    text_tokens = clip.tokenize(captions).to(device)

    with torch.no_grad():
        text_features = model.encode_text(text_tokens)
        text_features /= text_features.norm(dim=-1, keepdim=True)

    return text_features


def get_kth_highest_similarity_caption(k, model, device, preprocess, image_path,
                                       cached_text_features, captions):
    image = Image.open(image_path)
    image = preprocess(image).unsqueeze(0).to(device)

    with torch.no_grad():
        image_features = model.encode_image(image)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        similarities = (cached_text_features @ image_features.T).squeeze(0)
        .
        .
        .
```

Javascript Functions

1. **uploadImage() Function:**

   This function is triggered when a user attempts to upload an image. It sends a POST request to the /upload endpoint of the Flask server with the image data. Upon receiving a response from the server, it checks if the upload was successful: If successful, it displays the generated meme caption within an alert box. If there is an error (e.g., file format issue, no file selected), it displays an appropriate error message to the user.

2. **regenerateImage() Function:**

   This function makes a call to the /regenerate endpoint in order to get a new caption for the image on the user's request. On success, it shows the new caption in the message box.

3. **makeMemeImage() Function:**

   This function makes a GET request to the backend with the caption and the image filepath. On success, the API responds with a meme, which is the original image with the added caption.

4. **downloadImage() Function:**

   This function downloads the image to the user's device.

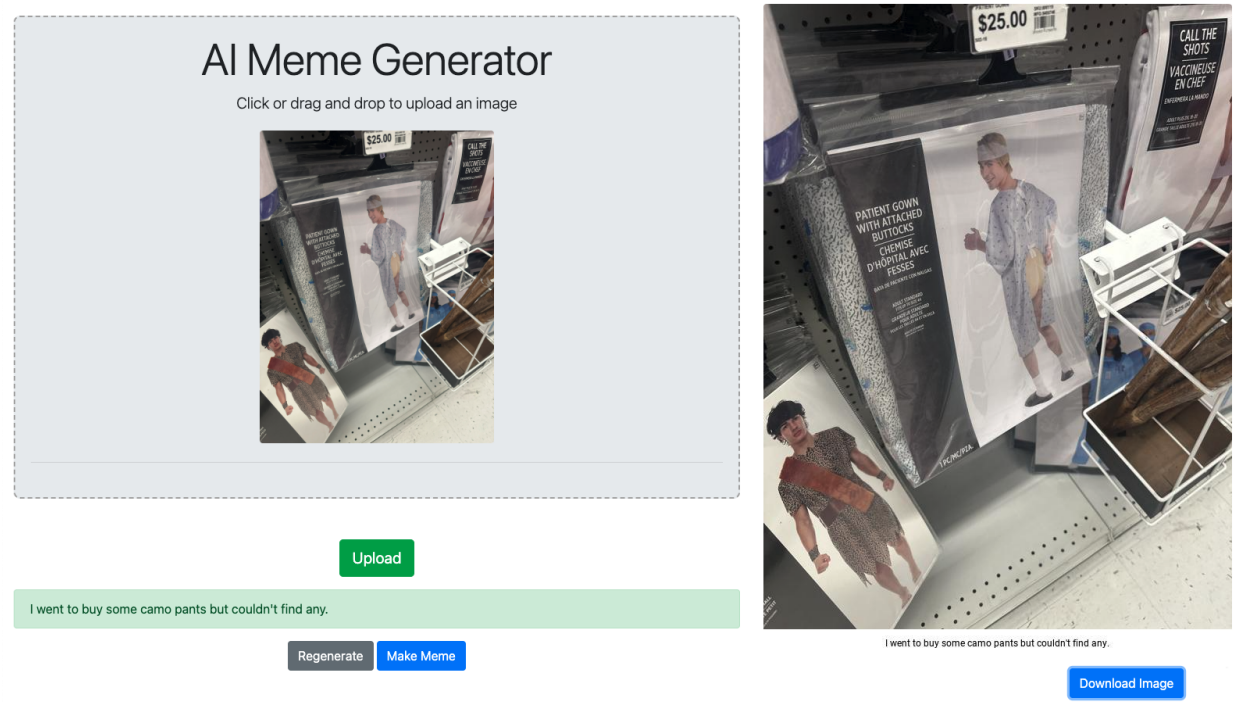
3 Discussion/Results: Learnings and Challenges

3.1 Learnings

We learned about integrating machine learning models in a web-based application. Particularly, we gained insights into using forms in http apis to upload blob data (image/videos). In addition, we gained insight into using caching to reduce latency in http apis and encoding

3.2 Demo

Below is an demo of our app displaying the buttons that correspond to the API's functionality, as well as the user interface for uploading the image, selecting a caption, and downloading the final meme.



3.3 Challenges

One of the significant challenges was optimizing the model for efficient real-time performance. To help speed up calculations, we cached CLIP encodings for 100 funny one-line captions – this way, we were able to decrease the number of calls.

4 Conclusions/Future Work: Reflections and Prospects

4.1 Future Work

In future iterations, we plan to:

- Enhance the variety of captions and themes
- Improve the model's efficiency and accuracy
- Explore dynamic caption generation based on user input or feedback

4.2 Reflections

This project has been a valuable experience in AI and web development, particularly in the niche of meme generation. We have successfully created a functional and user-friendly application that leverages advanced machine learning techniques.