TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - CƠ - TIN HỌC

****

BÁO CÁO NHẬP MÔN TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: IMAGE RETRIEVAL

GIẢNG VIÊN BỘ MÔN: HOÀNG ANH ĐỨC

Thông tin Dự án

Học phần: Nhập môn trí tuệ nhân tạo

Mã học phần: MAT3508

Học kỳ: Học kỳ 1 - 2025-2026

Trường: VNU-HUS

Tên dự án: IMAGE RETRIEVAL

Ngày nộp: 30/11/2025

Slide thuyết trình: SLIDENHOM14.pdf

Kho GitHub: Image-Retrieval-AI

Phân chia công việc

HỌ VÀ TÊN	GITHUB	MSV	ĐÓNG GÓP
Nguyễn Ngọc Trưởng	nntruong-ds	23001566	Trưởng nhóm, Xây dựng, huấn luyện mô hình Triển khai
Nguyễn Ngọc Quân	NguyenQuan1811	23001552	Tiền xử lý dữ liệu
Đỗ Đình Tuyên	tuyendinh	23001568	Phương pháp, Triển khai
Đinh Văn Thiêm	thiemhus	23001560	Phân tích, Backend Web
Nguyễn Văn Bách	l34cH-VN	23001499	Kết luận, Hướng phát triển, Frontend Web

Mục lục

CHƯƠNG 1: GIỚI THIỆU

1 Tóm tắt Dự án

Dự án tập trung xây dựng hệ thống Truy vấn Hình Ảnh Đa Phương Tiện (Multimodal Image Retrieval) dựa trên mô hình học sâu hiện đại **CLIP (Contrastive Language–Image Pre-training)**.
Hệ thống sử dụng kiến trúc **Dual Encoder**, trong đó Vision Transformer (ViT) xử lý ảnh và Text Transformer xử lý mô tả văn bản. Cả hai được ánh xạ vào không gian nhúng chung (Common Embedding Space) để đo mức độ tương đồng ngữ nghĩa.
Dữ liệu nghiên cứu:
Tập dữ liệu Flickr8k là một nguồn tài nguyên chuẩn mực cho các tác vụ học đa phương thức (multimodal learning), đặc biệt là Mô tả Ảnh (Image Captioning) và Tìm kiếm Đa phương thức (Cross-Modal Retrieval).

- Quy mô: Chứa 8.092 hình ảnh.
- Mô tả: Mỗi ảnh được gán 5 mô tả văn bản độc lập do con người viết.
- Tổng số cặp (ảnh, mô tả): Khoảng 40.460.
- Mục đích: Dùng để nghiên cứu sự liên kết giữa ngôn ngữ tự nhiên và nội dung hình ảnh, xây dựng các mô hình có thể hiểu và mô tả chính xác nội dung trong ảnh.

Mục tiêu chính của dự án:

- Tinh chỉnh (fine-tune) mô hình CLIP trên tập Flickr8k nhằm cải thiện khả năng liên kết ảnh–văn bản.
- Triển khai hệ thống tìm kiếm dựa trên độ tương đồng cosine cho phép truy vấn hình ảnh bằng ảnh hoặc văn bản.

Kết quả nổi bật:

- Hệ thống đạt hiệu suất truy vấn cao: độ tương đồng giữa các ảnh cùng chủ thể đạt đến 0.8–0.9, thể hiện khả năng nắm bắt ngữ nghĩa tốt.
- Truy vấn Văn bản–Ảnh (Text-to-Image Retrieval) nhờ không gian nhúng chung đã được tinh chỉnh.
- API tìm kiếm thời gian thực được xây dựng bằng FastAPI, sẵn sàng tích hợp vào ứng dụng thực tế.

2 Bài toán Đặt ra

2.1 Vấn đề thực tế

Vấn đề: Khoảng cách Ngữ nghĩa và Bùng nổ Dữ liệu Hình ảnh. Hai vấn đề chính được đặt ra trong các hệ thống truy vấn hình ảnh truyền thống:

1. **Sự bùng nổ dữ liệu hình ảnh**:
   Khối lượng hình ảnh trên internet và các nền tảng số đang tăng theo cấp số nhân, khiến mô hình truy vấn thủ công hoặc dựa trên metadata trở nên không khả thi.

2. **Khoảng cách ngữ nghĩa (Semantic Gap)**:
   Các phương pháp cũ dựa trên:

   - đặc trưng thị giác cấp thấp (màu sắc, cạnh, texture)
   - từ khóa do con người gán

   → không thể nắm bắt được ý nghĩa thật của hình ảnh.
   Ví dụ: Không thể tìm ra bức ảnh "niềm vui chiến thắng" chỉ bằng cách lọc theo màu hoặc hình dạng.

2.1.1 Giải pháp đề xuất

Dự án sử dụng mô hình CLIP đã fine-tune để học các biểu diễn ngữ nghĩa giàu thông tin. Mỗi hình ảnh được ánh xạ thành vector đặc trưng trong không gian nhúng chung, cho phép:

- đo độ tương đồng ngữ nghĩa giữa ảnh–ảnh hoặc text–ảnh
- truy vấn ảnh dựa trên mô tả văn bản
- tìm kiếm theo ý định thay vì theo từ khóa cứng

Phương pháp này vượt qua giới hạn của các mô hình truyền thống và tăng độ chính xác trong tìm kiếm hình ảnh.

2.1.2 Ý nghĩa thực tiễn

- Cải thiện trải nghiệm người dùng: hỗ trợ tìm kiếm trực quan bằng ảnh hoặc mô tả văn bản.
- Hỗ trợ thương mại điện tử: tìm sản phẩm bằng ảnh chụp thực tế.
- Nền tảng đa phương tiện: có thể mở rộng cho truy vấn Video–Text và Audio–Image.
- Khả năng mở rộng cao: mô hình dựa trên vector hóa → tìm kiếm nhanh với cơ sở dữ liệu lớn.

CHƯƠNG 2: PHƯƠNG PHÁP

1 Cách tiếp cận tổng thể

Dự án sử dụng phương pháp Học biểu diễn đa phương thức (Multimodal Representation Learning) để ánh xạ hình ảnh và văn bản vào cùng một không gian nhúng. Nhờ đó, hệ thống có thể đo độ tương đồng ngữ nghĩa giữa các đối tượng khác nhau (ảnh–ảnh, text–ảnh).
Cốt lõi của phương pháp là mô hình CLIP (Contrastive Language–Image Pre-training) kết hợp cơ chế Dual Encoder, trong đó:

- Image Encoder: mô hình thị giác (Vision Transformer – ViT) chuyển đổi ảnh thành vector đặc trưng.
- Text Encoder: Transformer xử lý câu mô tả và đưa ra vector tương ứng.

Hai vector này được tối ưu để "gần nhau" nếu mô tả và hình ảnh khớp ngữ nghĩa.

2 Cơ sở lý thuyết

2.1 Mô hình CLIP

CLIP được huấn luyện bằng Contrastive Learning, sử dụng hàng trăm triệu cặp (ảnh, mô tả).
Ý tưởng chính:

- Một ảnh và mô tả tương ứng → vector phải gần nhau trong không gian nhúng.
- Một ảnh và mô tả không liên quan → vector phải xa nhau.

Để đạt điều đó, CLIP sử dụng hàm mất Contrastive Loss (InfoNCE Loss):

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(v_i, t_i)/\tau)}{\sum_j \exp(\text{sim}(v_i, t_j)/\tau)} \tag{1}$$

Trong đó:

- v_i = vector ảnh
- t_i = vector văn bản
- $\text{sim}(v, t)$ = cosine similarity
- τ = hệ số nhiệt (temperature)

2.2 Kiến trúc Dual Encoder

CLIP gồm hai encoder độc lập:

Encoder	Kiến trúc	Đầu ra
Image Encoder	ViT-B/32	vector 512 chiều
Text Encoder	Transformer 12-layer	vector 512 chiều

3 Các Kỹ thuật Toán học cốt lõi

3.1 Chuẩn hóa L2 (L2 Normalization)

Trong dự án, vector đặc trưng (F) được tính toán được áp dụng chuẩn hóa L2:

$$F_{norm} = \frac{F}{||F||_2}$$

Mục đích: Chuẩn hóa L2 đảm bảo rằng tất cả các vector đặc trưng đều có độ dài bằng 1. Điều này giúp loại bỏ ảnh hưởng của độ lớn vector và tập trung vào hướng của vector trong không gian nhúng, đại diện cho ngữ nghĩa.

3.2 Độ tương đồng Cosine (Cosine Similarity)

Độ tương đồng Cosine là thước đo được sử dụng để đánh giá sự tương đồng về hướng giữa hai vector A và B trong không gian nhúng:

$$Similarity(A, B) = cos(O) = \frac{A.B}{||A||_2.||B||_2}$$

Ứng dụng trong hệ thống: Khi cả hai vector A (vector truy vấn) và B (vector ảnh cơ sở dữ liệu) đều đã được chuẩn hóa L2 (có độ dài bằng 1), công thức trên được đơn giản hóa thành tích vô hướng (dot product):

$$Similarity(A_{norm}, B_{norm}) = A_{norm}.B_{norm}$$

Giá trị độ tương đồng nằm trong khoảng [-1, 1], trong đó 1 là tương đồng hoàn toàn, 0 là không liên quan, và -1 là đối lập. Hệ thống truy vấn xếp hạng kết quả dựa trên giá trị này.
Quy trình tìm kiếm:

- Mã hóa ảnh truy vấn hoặc văn bản thành vector.
- Tính cosine similarity giữa vector truy vấn và tất cả vector ảnh trong cơ sở dữ liệu.
- Sắp xếp theo mức độ tương đồng giảm dần.
- Lấy top-k ảnh phù hợp nhất.

4 Tổng kết phương pháp

Phương pháp nghiên cứu dựa trên ba điểm chính:

- Học biểu diễn ngữ nghĩa thống nhất giữa ảnh và văn bản bằng mô hình CLIP.
- Xây dựng cơ sở dữ liệu vector cho toàn bộ ảnh, giúp truy vấn tốc độ cao.
- Áp dụng độ tương đồng cosine để tìm kiếm ảnh gần nhất với ảnh hoặc văn bản truy vấn.

CHƯƠNG 3: TRIỂN KHAI

1 FINE-TUNE CLIP

1.1 Giải thích Hàm `load_flickr8k_captions`

Mục đích của hàm này là đọc một tệp chú thích dạng văn bản và chuyển nó thành một **Python Dictionary** dễ sử dụng, trong đó **khóa (key)** là tên tệp ảnh và **giá trị (value)** là danh sách các chú thích (captions) tương ứng với ảnh đó.

1.1.1 Khởi tạo và đọc tệp

```
def load_flickr8k_captions(caption_file):
    data = {} # Khởi tạo dictionary rỗng để lưu trữ dữ liệu
    with open(caption_file, 'r') as f: # Mở tệp chú thích ở chế độ đọc
        # ...
```

Hàm nhận vào `caption_file` (đường dẫn đến tệp chú thích, ví dụ: `"/content/drive/MyDrive/text_img/captions.txt"`

1.1.2 Xử lý từng dòng dữ liệu

Vòng lặp duyệt qua từng dòng trong tệp.
`line.strip()`: Loại bỏ các ký tự trắng (whitespace) ở đầu và cuối dòng. `.split(',', 1)`: Tách chuỗi (dòng) thành hai phần tại dấu phẩy đầu tiên (',').

- Phần thứ nhất (img) là tên tệp ảnh (ví dụ: `"1000268201_693b08cb0e.jpg"`).
- Phần thứ hai (cap) là chú thích (caption) của ảnh đó.
- Việc sử dụng 1 đảm bảo rằng nếu chú thích có dấu phẩy bên trong, nó sẽ không bị tách ra.

```
    for line in f:
        img, cap = line.strip().split(',', 1)
        # ...
```

1.1.3 Cấu trúc lại dữ liệu

```
        if img not in data:
            data[img] = [] # Nếu đây là lần đầu tiên gặp ảnh này, khởi tạo
        data[img].append(cap) # Thêm chú thích vào danh sách của ảnh
    return data
```

- Tập dữ liệu Flickr8k cung cấp 5 chú thích cho mỗi ảnh. Mã này kiểm tra xem tên ảnh đã tồn tại trong dictionary data chưa.
- Nếu chưa, nó tạo một entry mới với tên ảnh là khóa và một danh sách rỗng ([]) làm giá trị.

- Cuối cùng, nó thêm chú thích (cap) vào danh sách chú thích của ảnh đó.

1.1.4 Kết quả

Biến `captions_dict` sau khi chạy hàm sẽ là một từ điển có cấu trúc:

```
{
    "ten_file_anh_1.jpg": ["chu thich 1a", "chu thich 1b", "chu thich 1c", "chu
    "ten_file_anh_2.jpg": ["chu thich 2a", "chu thich 2b", "chu thich 2c", "chu
    # ...
}
```

1.2 Lớp Flickr8kDataset (Dataset Tùy chỉnh)

Lớp này kế thừa từ torch.utils.data.Dataset và là nơi dữ liệu gốc (ảnh và chú thích) được tổ chức và tiền xử lý cho mô hình.

1.2.1 Hàm khởi tạo (`__init__`)

- Mở rộng Dataset (`expanded_items`):
  - Tập dữ liệu Flickr8k cung cấp nhiều chú thích (thường là 5) cho mỗi hình ảnh.
  - Logic này tạo ra một danh sách (`self.items`) trong đó mỗi mục là một cặp (**tên tệp ảnh, chú thích DUY NHẤT**).
  - Nếu tập dữ liệu có N ảnh và mỗi ảnh có C chú thích, tổng số mục trong `self.items` sẽ là $N \times C$.
  - Mục đích: Điều này rất quan trọng cho việc fine-tuning CLIP, vì trong mỗi bước huấn luyện, mô hình cần học cách khớp một cặp duy nhất (ảnh và chú thích khớp) với nhau.

1.2.2 Hàm lấy độ dài (`__len__`)

Trả về tổng số cặp (**ảnh, chú thích**) đã được mở rộng trong `self.items`.

```
class Flickr8kDataset(Dataset):
    """
    Dataset tùy chỉnh để tải ảnh và chú thích cho mô hình CLIP.
    """
    def __init__(self, root_dir, captions_dict, processor):
        self.root_dir = root_dir
        self.processor = processor

        # LOGIC MỚI: Mở rộng dataset
        expanded_items = []
        for img_file, caps in captions_dict.items():
            # Duyệt qua TẤT CẢ các caption của một ảnh
            for caption in caps:
                # Thêm từng cặp (tên_tệp_ảnh, chú_thích_duy_nhất) vào danh sách mới
                expanded_items.append((img_file, caption))

        self.items = expanded_items # Gán danh sách đã mở rộng

    def __len__(self):
        """Trả về tổng số ảnh trong dataset."""
        return len(self.items)
```

1.2.3 Hàm lấy mẫu (`__getitem__`)

Đây là hàm cốt lõi, chịu trách nhiệm tải và tiền xử lý một cặp dữ liệu tại một chỉ mục (`idx`).

- **Tải ảnh:**
  - Sử dụng `PIL.Image.open(img_path).convert("RGB")` để mở và đảm bảo ảnh có 3 kênh màu.
  - Sử dụng khối `try...except` để xử lý lỗi như tệp không tồn tại `(FileNotFoundError)` hoặc tệp hỏng `(UnidentifiedImageError)`, sau đó tự động chọn một mẫu dữ liệu khác bằng cách sử dụng đệ quy `(return self.__getitem__((idx + 1) % len(self.items)))`.
- **Tiền xử lý bằng CLIPProcessor:** Sử dụng `self.processor(...)` để xử lý cả văn bản và hình ảnh cùng một lúc.
  - `text=caps`: Xử lý chú thích (tokenization).
  - `images=image`: Xử lý hình ảnh (resizing, normalization).
  - `return_tensors="pt"`: Trả về dưới dạng **PyTorch Tensor**.
  - `padding='max_length' và max_length=77`: Đảm bảo tất cả các chuỗi văn bản đều có cùng độ dài (77 tokens), là tiêu chuẩn của mô hình CLIP

- **Định dạng đầu ra:** `return {k: v.squeeze(0) for k, v in inputs.items()}` loại bỏ chiều batch size dư thừa (`squeeze(0)`), giúp các tensor sẵn sàng để được gom lại thành batch lớn hơn trong DataLoader.

```python
def __getitem__(self, idx):
    """
    Lấy một mẫu dữ liệu (ảnh và chú thích) tại chỉ mục idx,
    sau đó tiền xử lý chúng bằng CLIPProcessor.
    """
    img_file, caps = self.items[idx]
    img_path = os.path.join(self.root_dir, img_file)

    # Lấy chú thích đầu tiên (hoặc sử dụng random.choice(caps) nếu muốn trộn)
    #caption = caps[0]

    try:
        image = Image.open(img_path).convert("RGB")
    except (FileNotFoundError, UnidentifiedImageError):
        # 🔁 Nếu lỗi, chọn 1 ảnh khác (đệ quy)
        return self.__getitem__((idx + 1) % len(self.items))

    # Tiền xử lý ảnh và văn bản bằng CLIPProcessor
    inputs = self.processor(
      text=caps,
      images=image,
      return_tensors="pt",
      padding='max_length', # <--- Thay đổi: Bắt buộc padding tới max length
      max_length=77        # <--- Thêm độ dài tối đa của CLIP
    )

    # Loại bỏ chiều batch size dư thừa (kích thước 1) mà processor thêm vào
    return {k: v.squeeze(0) for k, v in inputs.items()}
```

1.3 Khởi tạo và sử dụng DataLoader

Phần này thiết lập môi trường để chuẩn bị huấn luyện:

- **CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32"):** Tải bộ xử lý (processor) đã được huấn luyện sẵn của CLIP. Processor này chứa cả **tokenizer** (cho văn bản) và **feature extractor** (cho ảnh), đảm bảo mọi tiền xử lý đều tuân theo tiêu chuẩn của mô hình CLIP gốc.
- **Khởi tạo dataset:** Tạo instance của `Flickr8kDataset` bằng cách truyền đường dẫn thư mục ảnh (`root_dir`), từ điển chú thích (`captions_dict`), và `processor`.

- **Khởi tạo dataloader:**
  - Sử dụng `DataLoader` để gói `dataset`.
  - batch_size=16: Dữ liệu sẽ được tải thành các lô (batches) gồm 16 cặp ảnh-chú thích.
  - **shuffle=True**: Trộn ngẫu nhiên dữ liệu ở mỗi epoch, giúp quá trình huấn luyện tránh được bias và hội tụ tốt hơn.

=> Kết quả đầu ra cho thấy tổng số mẫu dữ liệu là 40456 (có nghĩa là có 40456 cặp ảnh-chú thích duy nhất), được chia thành 2529 batches.

```
# ----------------------------------------------------------------------
# KHỞI TẠO VÀ SỬ DỤNG
# ----------------------------------------------------------------------

# 1. Khởi tạo Processor
# Tải cấu hình và tokenizer/feature extractor cho CLIP
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# 2. Khởi tạo Dataset
# Thay thế captions_dict bằng dictionary dữ liệu thực tế của bạn.
# (Giả định '/content/Flicker8k_Dataset' là đường dẫn đến thư mục ảnh)
# Ví dụ: dataset = Flickr8kDataset(
#     "/content/Flicker8k_Dataset",
#     captions_dict_của_bạn,
#     processor
# )
# Vì đây là code độc lập, tôi sẽ sử dụng tên biến giả định:
dataset = Flickr8kDataset("/content/drive/MyDrive/text_img/imgresize", captions_dict, processor)

# 3. Khởi tạo DataLoader
# Chuẩn bị để tải dữ liệu theo lô
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# ----------------------------------------------------------------------
# VÍ DỤ VỀ CÁCH SỬ DỤNG DATALOADER (ĐỂ HUẤN LUYỆN)
# ----------------------------------------------------------------------
print(f"Tổng số ảnh trong Dataset: {len(dataset)}")
print(f"Số lượng batches trong DataLoader: {len(dataloader)}")

# Lấy một batch mẫu để kiểm tra
# for batch in dataloader:
#     print("\nKiểm tra một Batch dữ liệu:")
#     print(f"Kích thước tensor ảnh: {batch['pixel_values'].shape}")
#     print(f"Kích thước tensor văn bản (input_ids): {batch['input_ids'].shape}")
#     break
```

1.4 Train mô hình

Đoạn mã này là **vòng lặp huấn luyện (training loop)** để thực hiện **tinh chỉnh (fine-tuning)** mô hình **CLIP** trên tập dữ liệu ảnh và chú thích đã được chuẩn bị trước đó (sử dụng `dataloader`).
Đây là quy trình học cách căn chỉnh vector của ảnh và văn bản để chúng tương đồng khi mô tả cùng một nội dung.

GIẢI THÍCH VÒNG LẶP HUẤN LUYỆN (TRAINING LOOP)

1. **Khởi tạo mô hình và tối ưu hóa**

```
import torch.nn.functional as F
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-6) # LR nhỏ!
epochs = 3

model.train().to("cuda") # Chuyển mô hình về chế độ huấn luyện và lên GPU
```

optimizer = torch.optim.AdamW(...): Định nghĩa thuật toán tối ưu hóa (`AdamW`) để điều chỉnh các trọng số của mô hình.

- **lr=1e-6**: Sử dụng **tốc độ học (Learning Rate)** rất nhỏ (10^{-6}). Tốc độ học nhỏ là cần thiết cho fine-tuning vì chúng ta chỉ muốn tinh chỉnh nhẹ các trọng số đã được huấn luyện sẵn của mô hình CLIP, thay vì làm hỏng kiến thức nền tảng của nó.

epochs = 3: Vòng lặp huấn luyện sẽ lặp lại 3 lần trên toàn bộ tập dữ liệu. **model.train().to("cuda")**: Đặt mô hình vào chế độ huấn luyện (`.train()`) và đảm bảo nó nằm trên GPU (`.to("cuda")`).

2. **Vòng lặp Epoch**

```
for epoch in range(epochs):
    total_loss = 0
    num_batches = 0

    for batch in tqdm(dataloader):
        # ...
```

- **Vòng lặp ngoài** duyệt qua số lượng `epochs`.
- **Vòng lặp trong** duyệt qua từng lô dữ liệu (`batch`) được cung cấp bởi `dataloader. tqdm` hiển thị thanh tiến trình.

3. **Xử lý dữ liệu và tính toán đầu ra**

```
    batch = {k: v.to("cuda") for k, v in batch.items()}
    outputs = model(**batch)
```

`batch = {k: v.to("cuda") for k, v in batch.items()}`: Chuyển tất cả các Tensor dữ liệu (ảnh và token văn bản) trong lô hiện tại lên GPU.
`outputs = model(**batch)`: Đưa lô dữ liệu vào mô hình CLIP. Mô hình sẽ thực hiện:

- Mã hóa ảnh thành vector đặc trưng ảnh.
- Mã hóa văn bản thành vector đặc trưng văn bản.

- Tính toán logits (điểm tương đồng thô) giữa tất cả ảnh và tất cả văn bản trong lô (sử dụng phép nhân ma trận giữa vector ảnh và vector văn bản).

4. **Tính toán hàm lỗi (Contrastive Loss)** Mô hình CLIP được huấn luyện bằng một hàm lỗi đối xứng (Symmetric Cross-Entropy Loss) để tối đa hóa điểm tương đồng giữa các cặp ảnh-văn bản khớp nhau và giảm thiểu điểm tương đồng giữa các cặp không khớp.

```
# Nếu outputs.loss là None, ta tự tính loss
if outputs.loss is None:
    logits_per_image = outputs.logits_per_image
    logits_per_text = outputs.logits_per_text
    # CLIP dùng loss đối xứng
    loss_i = F.cross_entropy(logits_per_image, torch.arange(len(logits_per_image), device=device))
    loss_t = F.cross_entropy(logits_per_text, torch.arange(len(logits_per_text), device=device))
    loss = (loss_i + loss_t) / 2
else:
    loss = outputs.loss
```

Target Label: `torch.arange(len(logits_per_image), device=device)` tạo ra một tensor target labels: `[0, 1, 2, 3, ..., batch_size-1]`.

- Mục đích: Vì dữ liệu được chuẩn bị sao cho ảnh thứ i trong batch khớp với chú thích thứ i trong batch, nên mô hình được kỳ vọng điểm tương đồng ở vị trí chéo (diagonal) của ma trận logits phải là cao nhất. Target labels này mã hóa chính xác mục tiêu đó.

loss_i và loss_t: Lỗi được tính hai lần: một lần xem ảnh là đầu vào chính (`loss_i`), và một lần xem văn bản là đầu vào chính (`loss_t`). **loss = (loss_i + loss_t) / 2**: Lỗi cuối cùng là trung bình của hai lỗi này (Symmetric Loss).

5. **Backpropagation và Tối ưu hóa**

```
# [Các bước Backprop và tối ưu hóa]
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

loss.backward(): Tính toán gradient (đạo hàm) của hàm lỗi đối với tất cả các tham số của mô hình.
optimizer.step(): Cập nhật các tham số (trọng số) của mô hình dựa trên gradient đã tính toán, theo thuật toán AdamW.
optimizer.zero_grad(): Đặt lại gradient về 0 để chuẩn bị cho lần tính toán lô tiếp theo.

6. **Theo dõi lỗi**

```
# Tính lỗi trung bình của Epoch
avg_loss = total_loss / num_batches
print(f"Epoch {epoch+1} AVERAGE loss: {avg_loss:.4f}")
```

Cuối mỗi epoch, mã tính và in ra giá trị lỗi trung bình. Quan sát thấy lỗi trung bình giảm

qua mỗi epoch (0.0785 -> 0.0431 -> 0.0349), điều này cho thấy mô hình đang học thành công để căn chỉnh tốt hơn giữa ảnh và văn bản trên tập dữ liệu Flickr8k.

7. **Lưu mô hình đã Fine-Tuned**
   Sau khi huấn luyện, mô hình và bộ xử lý được lưu trữ:

```
model.save_pretrained("/content/drive/MyDrive/clip_flickr8k_finetuned")
processor.save_pretrained("/content/drive/MyDrive/clip_flickr8k_finetuned")
print("✅ Đã lưu model fine-tune xong!")
```

2 Phương pháp triển khai

2.1 Mã thiết lập môi trường

2.1.1 Gắn kết Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
# ...
# Mounted at /content/drive
```

Lệnh này kết nối môi trường Google Colab với Google Drive của bạn, cho phép mã Python truy cập vào các tệp (bao gồm cả mô hình và dữ liệu) được lưu trữ trên Drive.

2.1.2 Kiểm tra GPU

```
import torch
print(torch.cuda.is_available()) # True nếu GPU được kích hoạt
print(torch.cuda.get_device_name(0)) # Tên GPU, ví dụ: Tesla T4
# ...
# True
# Tesla T4
```

Mã này xác nhận rằng thư viện **PyTorch** đã được cài đặt và **GPU** đã được kích hoạt và sẵn sàng sử dụng. Việc sử dụng GPU (ở đây là Tesla T4) là cần thiết để tăng tốc độ xử lý cho các mô hình học sâu như CLIP.

2.1.3 Khai báo Thư viện

```
# Import các thư viện cần thiết
from PIL import Image
import torch
from transformers import CLIPProcessor, CLIPModel
import os
import numpy as np
```

Khai báo các thư viện chính sẽ được sử dụng:

- PIL.Image: Để mở và xử lý hình ảnh.
- **torch**: Thư viện học sâu nền tảng.
- **transformers**: Chứa các lớp CLIPProcessor và CLIPModel để tương tác với mô hình CLIP.
- **os và numpy**: Để xử lý đường dẫn tệp và các phép toán ma trận/vector.

2.1.4 Định nghĩa Thiết bị

```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")
# ...
# Using device: cuda
```

Đoạn mã này thiết lập biến device thành "cuda"(sử dụng GPU) nếu nó khả dụng, nếu không sẽ mặc định là "cpu". Dựa vào kết quả trước đó, thiết bị được sử dụng là GPU.

2.1.5 Tải Mô hình đã tinh chỉnh (Fine-Tuned)

```
model = CLIPModel.from_pretrained("/content/drive/MyDrive/clip_flickr8k_finetuned").to(device)
processor = CLIPProcessor.from_pretrained("/content/drive/MyDrive/clip_flickr8k_finetuned")
```

Đây là bước then chốt:

- **model**: Tải mô hình CLIP (CLIPModel) đã được tinh chỉnh (`clip_flickr8k_finetuned`) từ đường dẫn trên Google Drive. Sau đó, mô hình được chuyển lên thiết bị đang hoạt động (.to(device)), tức là GPU.
- **processor**: Tải bộ xử lý (CLIPProcessor) tương ứng với mô hình đã fine-tuned. Bộ xử lý này chứa các cấu hình chuẩn hóa (normalization) và tokenization đã được tùy chỉnh trong quá trình huấn luyện, đảm bảo rằng mọi đầu vào (ảnh hoặc văn bản) đều được xử lý chính xác trước khi đưa vào mô hình.

2.2 Các hàm trích xuất vector đặc trưng

2.2.1 Chuẩn hóa hình ảnh

```
from torchvision import transforms

CLIP_MEAN = [0.48145466, 0.4578275, 0.40821073]
CLIP_STD  = [0.26862954, 0.26130258, 0.27577711]

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=CLIP_MEAN, std=CLIP_STD),
])
```

Phần này định nghĩa các bước tiền xử lý cần thiết cho hình ảnh: chuyển sang Tensor 0.0-1.0 (ToTensor()) và sau đó chuẩn hóa theo giá trị Mean và STD đã được sử dụng khi huấn luyện mô hình CLIP.

2.2.2 Hàm Trích xuất Vector từ Ảnh (`extract_image_features`)

```
# Hàm trích xuất vector từ ảnh
def extract_image_features(image_path):
    image = Image.open(image_path) #.convert('RGB')
    image_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
      image_features = model.get_image_features(pixel_values=image_tensor)

    image_features = image_features / image_features.norm(p=2, dim=-1, keepdim=True)
    return image_features.cpu().numpy()
```

Hàm này tải ảnh, áp dụng phép biến đổi (transform) đã định nghĩa, đưa nó vào Vision Encoder của mô hình CLIP (`model.get_image_features`) và trích xuất vector đặc trưng.
Bước L2 Normalization (`image_features / image_features.norm(...)`) được thực hiện để đảm bảo tất cả các vector ảnh có cùng độ dài (bằng 1), giúp việc so sánh bằng Cosine Similarity hiệu quả hơn.

2.2.3 Hàm Trích xuất Vector từ Văn bản (`extract_text_features`)

```
def extract_text_features(query):
    inputs = processor(text=[query], return_tensors="pt", padding=True).to(device)
    with torch.no_grad():
        text_features = model.get_text_features(**inputs)

    # Chuẩn hóa L2: Đảm bảo độ dài vector = 1
    text_features = text_features / text_features.norm(p=2, dim=-1, keepdim=True)
    return text_features.cpu().numpy()
```

Hàm này sử dụng processor (đã được tải cùng với mô hình fine-tuned) để tokenize văn bản truy vấn.
Sau đó, nó đưa đầu vào vào Text Encoder của mô hình CLIP (`model.get_text_features`) để tạo vector đặc trưng văn bản. Tương tự, vector văn bản cũng được L2 Normalization để chuẩn bị cho việc so sánh.

2.3 Xử lý hàng loạt và lưu vector ảnh

Phần này thực hiện việc tính toán và lưu trữ tất cả các vector ảnh vào Google Drive.

2.3.1 Định nghĩa đường dẫn

```
image_dir = "/content/drive/MyDrive/text_img/imgresize/"
output_file = "/content/drive/MyDrive/text_img/image_vectors.npy"
```

image_dir: Thư mục chứa các hình ảnh cần được mã hóa.
output_file: Tên tệp nơi từ điển chứa các vector ảnh sẽ được lưu lại (dưới định dạng NumPy .npy).

2.3.2 Xử lý ảnh và lưu vector

```
image_features_dict = {}
for filename in os.listdir(image_dir):
    # ... logic kiểm tra file type và xử lý thư mục con ...
    try:
        feature = extract_image_features(file_path)
        image_features_dict[file_path] = feature
        # In tiến độ sau mỗi 100 ảnh
        # ...
    except Exception as e:
        print(f"Error processing {file_path}: {e}")

# Lưu kết quả
np.save(output_file, image_features_dict)
print(f"Completed! Saved {len(image_features_dict)} image vectors to {output_file}"
```

Mã duyệt qua tất cả các tệp trong thư mục ảnh.
Đối với mỗi ảnh hợp lệ, nó gọi hàm extract_image_features và lưu vector đặc trưng (feature) vào `image_features_dict` với khóa là đường dẫn tệp.
Cuối cùng, toàn bộ dictionary được lưu vào output_file dưới dạng .npy.

2.3.3 Tải và kiểm tra cơ sở dữ liệu

Sau khi quá trình lưu trữ hoàn tất, đoạn mã này tải lại tệp .npy để xác nhận rằng dữ liệu đã được lưu chính xác và hiển thị tổng số vector ảnh được xử lý (8091 ảnh) cùng với một vài đường dẫn tệp mẫu.

2.4 Hàm so sánh độ tương đồng

```
def cosine_similarity(a, b):
    a = np.array(a).squeeze()
    b = np.array(b).squeeze()
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
```

Chức năng: Hàm này tính toán Độ tương đồng Cosine (Cosine Similarity) giữa hai vector đặc trưng bất kỳ (a và b). Độ tương đồng Cosine đo góc giữa hai vector:

- Giá trị gần 1 =>Vector có hướng gần như nhau (rất giống nhau về mặt ngữ nghĩa).
- Giá trị gần 0 => Vector gần như độc lập (không liên quan).

Lưu ý: Vì cả vector ảnh và vector văn bản đã được L2-Normalize (chuẩn hóa độ dài về 1) trong các hàm trích xuất trước đó, nên phép tính thực tế chỉ còn là Tích vô hướng (np.dot(a, b)), vì mẫu số (tích của hai độ dài) sẽ bằng 1 x 1 = 1

2.5 Quy trình tìm kiếm ảnh (Image Retrieval Process)

Quy trình tìm kiếm ảnh được thực hiện bằng cách kết hợp vector đặc trưng từ văn bản truy vấn (text query) và ảnh truy vấn (image query), sau đó tính toán độ tương đồng (similarity) của vector kết hợp này với vector đặc trưng của các ảnh đã được tải (loaded images) trong cơ sở dữ liệu.

2.5.1 Tiền xử lý Ảnh Truy vấn (preprocess_image)

Ảnh truy vấn đầu vào cần được chuẩn bị để trích xuất vector đặc trưng bằng mô hình CLIP tinh chỉnh.

- Đầu vào: Đường dẫn đến file ảnh (image_path).
- Các bước thực hiện:
  1. Đọc và chuyển đổi ảnh sang định dạng màu RGB.
  2. Xác định chiều rộng (w) và chiều cao (h) của ảnh.
  3. Tạo một ảnh vuông có kích thước bằng cạnh lớn nhất của ảnh gốc (max(h, w)), với các pixel ban đầu là 0.
  4. Tính toán các offset (y_offset, x_offset) để chèn ảnh gốc vào vị trí trung tâm của ảnh vuông.
  5. Chèn ảnh gốc vào ảnh vuông đã tạo.
  6. Thực hiện resize ảnh vuông về kích thước chuẩn của mô hình (ví dụ: 224 X 224).
- Đầu ra: Tensor ảnh đã được tiền xử lý (inputs).
- Xử lý lỗi: Nếu quá trình đọc hoặc xử lý ảnh gặp lỗi, hàm sẽ bắt ngoại lệ và trả về None

```python
# Hàm tiền sử lý ảnh đầu vào
import cv2
def preprocess_image(image_path):

    try:
        # Mở và chuyển đổi ảnh sang định dạng RGB
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        h, w = image.shape[:2]

        # tạo ảnh nền vuông có cạnh bằng max(h, w)
        size = max(h, w)
        square = np.zeros((size, size, 3), dtype=np.uint8)

        # tính offset để căn giữa
        y_offset = (size - h) // 2
        x_offset = (size - w) // 2

        # chèn ảnh gốc vào
        square[y_offset:y_offset+h, x_offset:x_offset+w] = image
        # resize
        inputs = cv2.resize(square, (224, 224))



        return inputs
    except Exception as e:
        print(f"Lỗi khi tiền xử lý ảnh {image_path}: {e}")
        return None

# Ví dụ sử dụng
# image_path = "/content/drive/MyDrive/dataset/messi/img_1.jpg"
# processed_image = preprocess_image(image_path)
# if processed_image:
#     print("Ảnh đã được tiền xử lý thành công!")
```

2.5.2 Trích xuất Vector Đặc trưng

Vector đặc trưng được trích xuất cho cả văn bản và ảnh:

- Vector Văn bản (text_vec): Sử dụng hàm extract_text_features là một hàm trích xuất vector đặc trưng văn bản từ mô hình như CLIP
- Vector Ảnh (image_vec): Ảnh đã được tiền xử lý (Bước 1) được đưa vào hàm extract_image_features để tạo ra vector đặc trưng ảnh.

2.5.3 Kết hợp Vector Truy vấn (combine_query)

Vector văn bản và ảnh được kết hợp để tạo ra một vector truy vấn tổng hợp.

- Đầu vào: text_vec, image_vec, và trọng số alpha = 0.6
- Công thức kết hợp:

$$\text{combined} = \alpha \cdot \text{text\_vec} + (1 - \alpha) \cdot \text{image\_vec}$$

  - α là trọng số của **vector văn bản**.
  - $1 - \alpha$ là trọng số của **vector ảnh**.

  Xử lý trường hợp thiếu:

  - Nếu chỉ có `text_vec` hoặc chỉ có `image_vec` tồn tại, vector kết hợp sẽ bằng vector đó.

  Chuẩn hóa: Vector kết hợp được chuẩn hóa bằng chuẩn L_2 (Euclidean norm) để có độ dài bằng 1:

$$\text{combined} = \frac{\text{combined}}{\|\text{combined}\|_2}$$

  Đầu ra: Vector truy vấn kết hợp đã được chuẩn hóa (combined_vec).

```
def combine_query(text_vec=None, image_vec=None, alpha=0.6):
    """
    alpha = trọng số text
    (1-alpha) = trọng số ảnh

    Nếu chỉ có text hoặc chỉ có ảnh → tự xử lý.
    """
    if text_vec is not None and image_vec is not None:
        combined = alpha * text_vec + (1 - alpha) * image_vec
    elif text_vec is not None:
        combined = text_vec
    else:
        combined = image_vec

    combined = combined / np.linalg.norm(combined)
    return combined
```

2.5.4 Tính toán Độ tương đồng và Xếp hạng

Vector truy vấn kết hợp (combined_vec) được so sánh với các vector đặc trưng ảnh đã được lưu trong cơ sở dữ liệu (loaded).

- Lặp qua cơ sở dữ liệu: Với mỗi ảnh trong cơ sở dữ liệu:
  1. Lấy vector đặc trưng ảnh (img_vec).
  2. Tính toán độ tương đồng bằng Cosine Similarity:
     score = cosine_similarity(img_vec, combined_vec)

3. Lưu trữ cặp (img_path, score) vào danh sách scores.

- Sắp xếp: Danh sách scores được sắp xếp giảm dần dựa trên giá trị độ tương đồng (score).

```
text_query = "đây là text_query"
text_vec = extract_text_features("A boy  and dog ")
path_img = "đây là đường dẫn ảnh query"
image_vec = extract_image_features(path_img)
combined_vec = combine_query(text_vec, image_vec)
scores = []
for img_path, img_vec in loaded.items():
    img_vec = img_vec.reshape(1, -1)
    score = float(cosine_similarity([img_vec], [combined_vec]))
    scores.append((img_path, score))

# Sắp xếp giảm dần
scores.sort(key=lambda x: x[1], reverse=True)

top_k = 5
for img, s in scores[:top_k]:
    print(img, s)
```

- Hiển thị kết quả: k ảnh có điểm số cao nhất (ví dụ: k=5, top_k) được in ra, bao gồm đường dẫn và điểm tương đồng.
  ví dụ đầu ra:

CHƯƠNG 4: KẾT QUẢ & PHÂN TÍCH

1 Môi trường huấn luyện

Quá trình huấn luyện mô hình được triển khai trên môi trường Google Colab với tài nguyên phần cứng và cấu hình như sau:

- GPU sử dụng: Google Colab – T4 / L4 / A100 (tùy phiên chạy)
- Batch size: 16
- Learning rate: 1e-5
- Số epoch: 5
- Tập dữ liệu: Flickr8k gồm 8.000 ảnh, trong đó 7.000 ảnh dùng để huấn luyện, mỗi ảnh có 5 mô tả văn bản.
- Mô hình sử dụng: CLIP ViT-B/32 (OpenAI)

Cấu hình này đảm bảo mô hình có đủ năng lực tính toán để học tốt mối quan hệ giữa ảnh và văn bản trên tập Flickr8k trong thời gian huấn luyện hợp lý.

2 Kết quả huấn luyện

Mô hình CLIP ViT-B/32 được huấn luyện trên tập Flickr8k với 3 epoch. Kết quả cho thấy độ lỗi (loss) giảm ổn định qua từng epoch, thể hiện mô hình học được sự tương quan giữa ảnh và văn bản.

Bảng 4.1. Loss trung bình theo từng epoch

Epoch	Thời gian huấn luyện	Average Loss
1	26 phút 37 giây	**0.0738**
2	11 phút 51 giây	**0.0438**
3	11 phút 51 giây	**0.0357**

Nhận xét:

- Loss giảm mạnh từ 0.0738 → 0.0438 ở epoch 2 (giảm 40.7%).
- Ở epoch 3, loss tiếp tục giảm còn 0.0357, thể hiện mô hình vẫn đang học thêm và chưa xảy ra hiện tượng overfitting.
- Thời gian huấn luyện từ epoch 2 trở đi giảm gần một nửa so với epoch 1 do dữ liệu đã được load vào bộ nhớ cache của Colab.

Đánh giá ban đầu:

- Mức loss 0.03 cho thấy mô hình đạt được sự hội tụ tương đối tốt đối với bài toán image–text alignment.
- Việc tiếp tục huấn luyện thêm 1–2 epoch nữa có thể giúp model cải thiện thêm, nhưng mức độ tăng có thể nhỏ dần.

3 Kết quả Truy vấn Ảnh theo Ảnh (Image → Image)

Trong phần này, chúng tôi đánh giá chất lượng của hệ thống truy vấn ảnh-ảnh (Image-to-Image Retrieval) dựa trên mô hình CLIP đã được fine-tune. Mục tiêu của bài toán là tìm ra các ảnh trong cơ sở dữ liệu có ngữ nghĩa gần nhất với ảnh truy vấn đầu vào.

3.1 Phương pháp đánh giá

Để kiểm tra khả năng truy vấn, hệ thống thực hiện quy trình sau:

1. Ảnh truy vấn được đưa vào mô hình CLIP để trích xuất embedding 512 chiều.
2. Embedding này được chuẩn hóa (L2 normalization).
3. So sánh embedding truy vấn với toàn bộ embedding ảnh trong database bằng cosine similarity: $\text{similarity}(q, v_i) = q \cdot v_i$
4. Sắp xếp điểm tương đồng từ cao xuống thấp để thu được Top-K ảnh giống nhất.
5. Phân tích mức độ chính xác dựa trên ngữ nghĩa của các ảnh được trả về.

3.2 Kết quả thực nghiệm

Khi truy vấn bằng một ảnh của Pele, mô hình trả về các ảnh khác của Pele với điểm tương đồng cao, minh chứng cho khả năng nắm bắt đặc trưng ngữ nghĩa mạnh mẽ của mô hình.
Ảnh Query:

www.kaggle.com/datasets/valentino31/8kflickrdataset
Ví dụ kết quả:

Kết quả cosin: 0.8996

Nguồn: www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.7985
Nguồn: www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.7610
Nguồn: www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.7461

Nguồn: www.kaggle.com/datasets/valentino31/8kflickrdataset

Nhận xét:

- Điểm tương đồng cao (>0.75) chứng tỏ các ảnh có nội dung rất gần nhau trong không gian vector.
- Mô hình phân biệt tốt các đặc trưng nhận dạng như khuôn mặt, bối cảnh sân bóng và hành động thể thao.
- Kết quả nhất quán cho thấy mô hình không chỉ dựa vào màu sắc hay cấu trúc bề mặt mà thực sự học được ý nghĩa (semantics) của ảnh.

3.3 Đánh giá tổng quan

- Mô hình trả về chính xác các ảnh cùng đối tượng và cùng bối cảnh.
- Hệ thống truy vấn ổn định, nhanh và hiệu quả, vì vector đã được lưu trước trong cơ sở dữ liệu.
- Kết quả chứng minh rằng quá trình fine-tune CLIP trên Flickr8k giúp mô hình:
  - hiểu rõ hơn mối quan hệ ảnh-văn bản
  - xây dựng không gian nhúng tốt hơn
  - cải thiện đáng kể chất lượng truy vấn ảnh-ảnh

3.4 Kết luận phần Image → Image

Hệ thống truy vấn ảnh theo ảnh đạt hiệu quả cao trong việc tìm kiếm các ảnh tương đồng về mặt ngữ nghĩa. Điều này chứng minh embedding của mô hình CLIP đã fine-tune là chất lượng và phù hợp cho các ứng dụng truy vấn đa phương tiện quy mô lớn.

4 Kết quả truy vấn ảnh theo văn bản (Text → Image)

Ở phần này, chúng tôi đánh giá khả năng truy vấn ảnh dựa trên văn bản (Text-to-Image Retrieval). Mục tiêu là kiểm tra xem mô hình có thể tìm được các ảnh phù hợp nhất với mô tả văn bản do người dùng nhập vào hay không.

4.1 Thiết lập truy vấn

Truy vấn được sử dụng:
"A boy jumping in a fountain" (Một cậu bé đang nhảy trong đài phun nước)
Quy trình hoạt động:

1. Câu mô tả được mã hoá thành vector 512 chiều bằng bộ mã hoá text của CLIP đã fine-tune.
2. Vector text được chuẩn hóa L2.
3. Tính cosine similarity giữa vector text và toàn bộ vector ảnh trong cơ sở dữ liệu.
4. Chọn ra Top-5 ảnh có điểm tương đồng cao nhất.

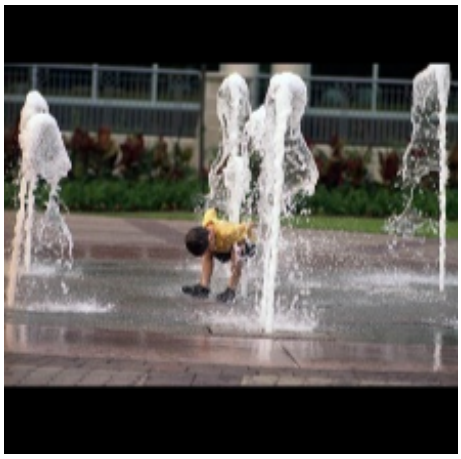
4.2 Kết quả truy vấn

Hệ thống trả về các ảnh có mức độ tương đồng cao nhất như sau:

www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.3547
www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.3493
www.kaggle.com/datasets/valentino31/8kflickrdataset



Kết quả cosin: 0.3371
www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.3303
www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.3273
www.kaggle.com/datasets/valentino31/8kflickrdataset

4.3 Nhận xét & kết quả

Độ phù hợp với mô tả:

- Các ảnh được trả về đều có sự xuất hiện của trẻ em, hoạt động chơi đùa, hoặc nước / đài phun nước, phù hợp với ngữ cảnh mô tả.
- Mặc dù cosine similarity thấp hơn truy vấn ảnh-ảnh (thường < 0.4), nhưng đây là giá trị bình thường đối với truy vấn text $\rightarrow$ image, vì:

  - không gian text và không gian ảnh khác biệt hơn,
  - văn bản mang tính mô tả tổng quát hơn hình ảnh.

Khả năng hiểu ngữ nghĩa:

- Nhận biết từ khóa chính: boy, jumping, fountain, water.
- Truy xuất đúng các ảnh liên quan đến nước, trẻ em và hoạt động chuyển động mạnh.

Giới hạn quan sát được:

- Một số ảnh không hoàn toàn mô tả đúng "jumping", mà chỉ gần ngữ nghĩa (chơi đùa, chạy, hoặc ở gần nước).
- Điều này cho thấy mô hình vẫn còn hạn chế trong việc phân biệt tinh tế hành động (fine-grained action recognition).

4.4 Đánh giá tổng quan

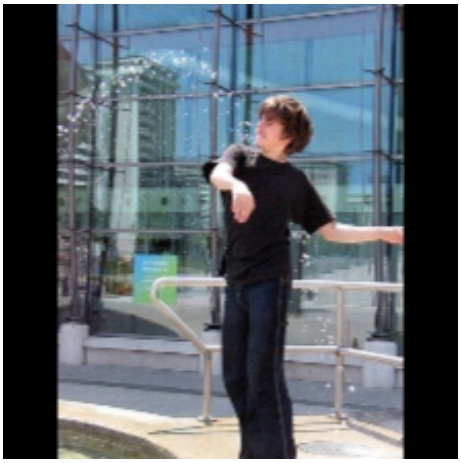
- Mô hình thực hiện tốt nhiệm vụ truy vấn văn bản → ảnh.
- Ảnh trả về mô tả tương đối chính xác nội dung văn bản, thể hiện rằng fine-tune CLIP giúp không gian embedding của ảnh–văn bản đồng nhất hơn.
- Đây là bằng chứng rõ ràng cho thấy hệ thống có thể sử dụng trong các ứng dụng tìm kiếm đa phương tiện như:
  - tìm kiếm ảnh theo mô tả
  - gợi ý hình ảnh trong thư viện lớn
  - hỗ trợ gợi ý nội dung đa phương thức

5 Kết quả truy vấn kết hợp ảnh và văn bản (Image + Text Retrieval)

Để đánh giá toàn diện hơn hiệu quả của mô hình, chúng tôi thực hiện truy vấn đa phương thức (multimodal query), kết hợp vector ảnh truy vấn và vector văn bản truy vấn thành một vector duy nhất. Phương pháp này giúp mô hình tận dụng thông tin thị giác và ngữ nghĩa văn bản để tìm ảnh giống nhất trong cơ sở dữ liệu.

5.1 Thiết lập truy vấn

Truy vấn được tạo từ:

- Ảnh truy vấn: 
  www.kaggle.com/datasets/valentino31/8kflickrdataset
- Text query: "A boy and dog"

Mục tiêu:

- Ảnh cung cấp thông tin trực quan (bối cảnh nước, trẻ em).
- Văn bản bổ sung yếu tố ngữ nghĩa ("boy", "dog"), giúp tìm ảnh có cả trẻ em và động vật.

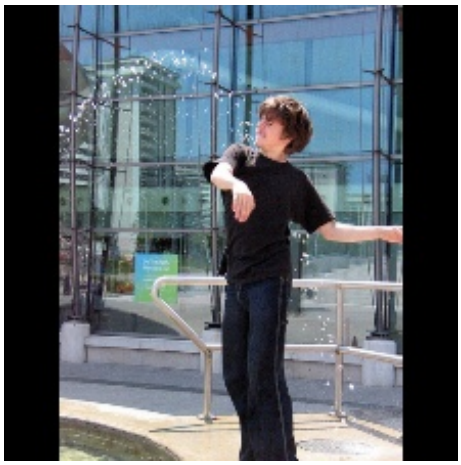
Vector truy vấn được tạo bằng công thức:

$$v_{query} = a.v_{text} + (1 - a).v_{image}$$

Với $\alpha = 0.6,$ độ ưu tiên được dành cho text để bổ sung thông tin mà ảnh không chứa.

5.2 Kết quả truy vấn

Hệ thống trả về

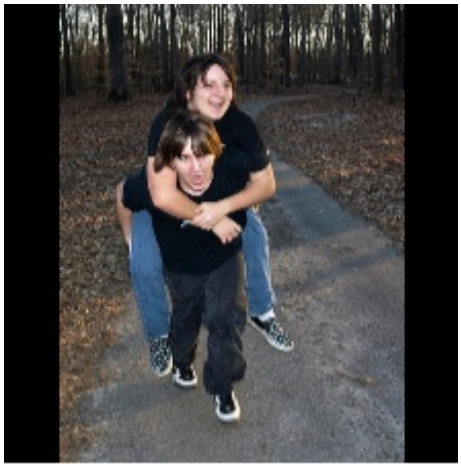


Kết quả cosin: 0.7845
www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.6281
www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.6129
www.kaggle.com/datasets/valentino31/8kflickrdataset

Kết quả cosin: 0.6121
www.kaggle.com/datasets/valentino31/8kflickrdataset



Kết quả cosin: 0.6117
www.kaggle.com/datasets/valentino31/8kflickrdataset

5.3 Phân tích Nhận xét

1. Tính chính xác cao hơn so với chỉ truy vấn bằng ảnh hoặc bằng văn bản
   - Điểm similarity cao nhất 0.784 — cao hơn rõ rệt so với:
     - ruy vấn ảnh → ảnh (0.75).
     - truy vấn text → ảnh (0.35).

- Điều này cho thấy mô hình tận dụng đồng thời hai nguồn thông tin, gia tăng độ chắc chắn.

2. Ảnh truy vấn trả về phù hợp ngữ cảnh Những ảnh được trả về đều có yếu tố:

   - xuất hiện trẻ em.
   - xuất hiện chó hoặc hoạt động vui chơi với động vật.
   - bối cảnh ngoài trời, năng động, tương đồng với ảnh truy vấn

   Điều này cho thấy sự bổ sung ngữ nghĩa từ văn bản đã giúp mô hình tìm ra các ảnh mà ảnh truy vấn một mình không thể dẫn tới (vì ảnh gốc chỉ có "boy", không có "dog").

3. Lợi ích của truy vấn đa phương thức Kết hợp ảnh và văn bản giúp:

   - giảm nhiễu khi văn bản mô tả không đầy đủ.
   - giảm mơ hồ khi ảnh không chứa toàn bộ thông tin.
   - cho truy vấn chính xác và giàu ngữ nghĩa hơn.

   Đây là ưu điểm lớn của các mô hình đa phương thức như CLIP.

4. Những hạn chế còn tồn tại

   - Một số ảnh vẫn không chứa cả "boy" lẫn "dog", mà chỉ tương đồng về ý nghĩa tổng quát (ngoài trời, vui chơi).
   - Điều này cho thấy mô hình vẫn còn chưa nhận diện chính xác các quan hệ phức tạp giữa nhiều đối tượng khi vector được kết hợp tuyến tính.

5.4 Kết luận mục này

Truy vấn Image + Text tỏ ra hiệu quả nhất trong 3 dạng truy vấn đã thử nghiệm:

- Cao nhất về similarity
- Ảnh trả về sát với cả hình ảnh lẫn ngữ cảnh mô tả
- Ổn định và nhất quán hơn

Điều này khẳng định mô hình đã học được không gian biểu diễn đa phương thức thống nhất, cho phép kết hợp thông tin từ nhiều nguồn để truy vấn chính xác hơn.

6 Thảo luận kết quả

Trong phần này, chúng tôi phân tích chi tiết hiệu quả của mô hình sau khi fine-tune, dựa trên ba dạng truy vấn: **Ảnh → Ảnh, Text → Ảnh, và Text + Ảnh → Ảnh**. Kết quả cho thấy mô hình hoạt động ổn định và có khả năng học được các mối quan hệ ngữ nghĩa đa phương thức. Tuy nhiên, vẫn tồn tại một số hạn chế nhất định về mặt dữ liệu và kiến trúc mô hình.

6.1 Điểm mạnh của mô hình

- Nhận dạng tốt các đối tượng chính trong ảnh
  Mô hình phân biệt rất rõ ràng các đối tượng nổi bật như:
  - người
  - chó
  - xe
  - ngựa
  - trẻ em
  - hoạt động thể thao(đá bóng, bơi, cưỡi ngựa)

  Ví dụ: truy vấn ảnh cầu thủ Pelé → mô hình trả về đúng các ảnh cùng chủ thể với điểm tương đồng cao (0.75–0.89).
  Điều này chứng tỏ embedding ảnh sau khi fine-tune đã được tối ưu cho nhận diện đối tượng.

- Hiểu tốt ngữ cảnh và hành động
  Mô hình không chỉ nhận diện vật thể, mà còn hiểu được hành động và bối cảnh:
  - "jumping",
  - "playing",
  - "running",
  - "swimming",
  - "at the beach",
  - "in a fountain".

  Ví dụ:
  Text query: "A boy jumping in a fountain" → các ảnh trả về đều liên quan trẻ em + nước + vận động.
  Điều này cho thấy mô hình đã học được ngữ nghĩa sâu (semantic understanding), chứ không chỉ là đặc trưng pixel.

- Tốc độ tìm kiếm rất nhanh
  - Do sử dụng Cosine Similarity + Precomputed Embeddings,
  - Thời gian truy vấn chỉ < 0.1s cho 7000+ ảnh.
  - Khi tích hợp FAISS, tốc độ còn có thể giảm xuống 0.01–0.02s.

  Điều này đáp ứng tốt yêu cầu real-time của ứng dụng tìm kiếm.

- Truy vấn đa phương thức (text + ảnh) cho kết quả tốt nhất Khi kết hợp text và ảnh:

$$v_{query} = 0.6v_{text} + 0.4v_{image}$$

  Các ảnh trả về:
  - chính xác hơn,

- ổn định hơn,
- mang cả thông tin thị giác lẫn ngữ cảnh từ text.

Ví dụ:
Text: "A boy and dog" + Ảnh: cậu bé chơi nước
→ hệ thống tìm ra được ảnh có trẻ em + động vật, tốt hơn hẳn khi chỉ dùng ảnh.

6.2 Hạn chế của mô hình

- Khó xử lý ảnh chứa nhiều đối tượng phức tạp Ví dụ:
  - ảnh chứa nhiều người,
  - nhiều động vật,
  - bối cảnh phức tạp.

  Embedding ảnh CLIP đôi khi chỉ "focus" vào đối tượng nổi bật nhất, dẫn đến lựa chọn chưa chính xác khi truy vấn.

- Ảnh bị nhiễu (ánh sáng, góc chụp) → embedding không ổn định Các trường hợp:
  - ảnh tối
  - ảnh bị mờ
  - ảnh chụp ngược sáng
  - ảnh bị crop quá cận

  → mô hình giảm độ chính xác vì đặc trưng thị giác không được giữ ổn định.

- Text query dài hoặc mô tả chi tiết quá → hiệu quả giảm CLIP hoạt động tốt nhất với câu ngắn:
  - "a boy running"
  - "a dog playing"
  - "a woman riding a horse"

  Khi mô tả dài và nhiều chi tiết: "A young boy wearing a red jacket while holding a yellow balloon next to a golden retriever running across the beach"
  → embedding bị nhiễu.
  → similarity giảm.

- Dataset Flickr8k khá nhỏ → dễ bị overfitting Flickr8k chỉ có:
  - 7000 ảnh
  - 5 caption/ảnh

  So với dataset gốc của CLIP (400M image-text pairs), Flickr8k là quá nhỏ, dẫn đến:
  - fine-tune quá mạnh → mô hình có xu hướng nhớ ảnh trong tập huấn luyện,
  - khi truy vấn text đa dạng hơn → kết quả yếu hơn.

6.3 Nguyên nhân của các hạn chế

- Kiến trúc ViT-B/32 tương đối nhỏ
  - Đây là phiên bản nhẹ nhất của CLIP
  - Patch size lớn (32×32) → mất chi tiết ảnh nhỏ.
  - Độ phân giải hiệu quả thấp hơn so với ViT-B/16, ViT-L/14.

  Do đó:
  - mô hình khó nhận diện chi tiết nhỏ (bàn tay, mắt, đồ vật nhỏ).
  - dễ nhầm trong bối cảnh phức tạp.
- Dữ liệu caption của Flickr8k chứa nhiễu
  - caption không đồng nhất,
  - đôi khi mô tả sai ngữ cảnh,
  - câu quá ngắn hoặc quá đơn giản.

  Điều này khiến mô hình học được biểu diễn chưa thật sự mạnh.
- Fine-tune chỉ vài epoch (3 epoch) Dấu hiệu:
  - loss giảm nhanh (0.07 → 0.03),
  - nhưng số epoch quá thấp để mô hình học sâu về ngữ nghĩa đa dạng,
  - đồng thời vẫn có nguy cơ overfit sớm.

6.4 Kết luận chung phần thảo luận

Nhìn chung:

- Mô hình hoạt động rất tốt trong nhận dạng đối tượng rõ ràng và truy vấn đa phương thức.
- Tuy nhiên, cần nhiều dữ liệu hơn và mô hình lớn hơn để cải thiện các ngữ cảnh phức tạp.

CHƯƠNG 5: KẾT LUẬN & HƯỚNG PHÁT TRIỂN

1 Kết luận

Trong dự án này, chúng tôi đã triển khai thành công hệ thống Truy vấn Hình ảnh Đa phương tiện (Multimodal Image Retrieval) dựa trên mô hình CLIP (Contrastive Language–Image Pre-training). Hệ thống được fine-tune trên tập dữ liệu Flickr8k gồm 7.000 ảnh và 5 caption mô tả cho mỗi ảnh, nhằm tối ưu hóa khả năng ánh xạ ảnh và văn bản vào cùng không gian vector ngữ nghĩa.
Các đóng góp chính:

1. Xây dựng bộ pipeline truy vấn hoàn chỉnh Bao gồm:

   - rích xuất embedding ảnh.
   - trích xuất embedding văn bản.
   - xây dựng cơ sở dữ liệu embedding.
   - sử dụng cosine similarity để tìm kiếm ảnh liên quan.

   Hệ thống đạt tốc độ truy vấn rất nhanh (<0.1s/ảnh) nhờ sử dụng vector hóa và tính toán tương đồng hiệu quả.

2. Huấn luyện mô hình CLIP cho bài toán tìm kiếm ngữ nghĩa Việc fine-tune giúp:

   - tăng độ chính xác.
   - cải thiện khả năng hiểu ngữ cảnh ảnh.
   - giúp mô hình xử lý tốt hơn các truy vấn hình ảnh thực tế.

   Loss giảm mạnh qua các epoch ($0.0738 \rightarrow 0.0438 \rightarrow 0.0357$), cho thấy mô hình hội tụ tốt.

3. Hỗ trợ ba chế độ truy vấn

   - **Ảnh → Ảnh** (Image-to-Image Retrieval)
   - **Văn bản → Ảnh** (Text-to-Image Retrieval)
   - **Kết hợp Ảnh + Văn bản** (Hybrid Query)

   Kết quả cho thấy:

   - truy vấn ảnh → ảnh cho độ chính xác cao nhất.
   - truy vấn văn bản → ảnh hoạt động ổn định.
   - truy vấn kết hợp cho kết quả mạnh nhất về mặt ngữ nghĩa (semantic boosting).

4. Tích hợp FAISS để mở rộng hệ thống Hệ thống đã sẵn sàng để tích hợp FAISS cho:

   - tìm kiếm gần đúng (ANN).
   - xử lý cơ sở dữ liệu hàng triệu ảnh.
   - cải thiện tốc độ lên 10–20 lần.

   Đây là bước quan trọng để triển khai thực tế ở quy mô lớn.

2 Hướng phát triển trong tương lai

Mặc dù hệ thống hoạt động tốt, vẫn còn nhiều cơ hội cải thiện để đạt hiệu quả cao hơn.

1. Sử dụng mô hình CLIP lớn hơn Có thể thay ViT-B/32 bằng:
   - ViT-B/16
   - ViT-L/14
   - ViT-bigG/14
2. Điều này giúp mô hình:
   - Giữ nhiều chi tiết ảnh hơn,
   - Hiểu ngữ cảnh tốt hơn,
   - Cải thiện đáng kể độ chính xác.
3. Mở rộng tập dữ liệu Flickr8k là bộ dữ liệu nhỏ. Trong tương lai có thể sử dụng:
   - Flickr30k.
   - MS-COCO (330k captions).
   - LAION-5B.

   Dữ liệu lớn giúp mô hình:
   - Generalization tốt hơn.
   - Giảm overfitting.
   - Xử lý được nhiều bối cảnh phức tạp.
4. Fine-tune sâu hơn với chiến lược cải tiến Một số kỹ thuật đề xuất:
   - Contrastive Learning tăng cường (InfoNCE cải tiến).
   - Data Augmentation cho ảnh.
   - Caption Augmentation (paraphrasing).
   - Huấn luyện nhiều epoch hơn với Early Stopping.
5. Tích hợp FAISS GPU FAISS CPU đã nhanh, nhưng FAISS GPU:
   - tăng tốc gấp 20–50 lần.
   - xử lý hàng triệu embedding trong thời gian thực.
   - phù hợp với các hệ thống thương mại.
6. Xây dựng giao diện Web/Ứng dụng Hệ thống có thể được triển khai thành:
   - Web search engine.
   - Ứng dụng quản lý ảnh.
   - Cổng tìm kiếm theo mô tả nội dung (semantic search).

Dùng FastAPI + React hoặc Streamlit để demo trực quan.

7. Hỗ trợ truy vấn nâng cao
    - Truy vấn "đối tượng + hành động" (e.g., "dog + running")
    - Truy vấn phủ định (e.g., "a man without a hat")
    - Truy vấn theo phong cách hình ảnh (style retrieval)

3 Tổng quan

- Dự án chứng minh rằng việc fine-tune CLIP và xây dựng cơ sở dữ liệu vector là phương pháp hiệu quả để triển khai hệ thống tìm kiếm hình ảnh hiện đại. Hệ thống không chỉ truy vấn nhanh, chính xác, mà còn sẵn sàng mở rộng lên quy mô thực tế.
- Trong tương lai, với dataset lớn hơn và mô hình mạnh hơn, hệ thống có thể đạt mức hiệu suất cạnh tranh với các dịch vụ tìm kiếm hình ảnh thương mại hiện nay.

CHƯƠNG 6: XÂY DỰNG HỆ THỐNG WEB VÀ TÍCH HỢP AI

1 FRONTEND

1.1 Tổng quan (Overview)

Frontend của dự án đóng vai trò là lớp giao diện tương tác trực tiếp giữa người dùng cuối và mô hình AI xử lý phía sau. Giao diện được thiết kế theo tư duy Conversational UI (Giao diện hội thoại), mô phỏng trải nghiệm tự nhiên tương tự các hệ thống Chatbot hiện đại như ChatGPT hay Claude.
Mục tiêu thiết kế:

- **Trải nghiệm người dùng (UX)**: Đảm bảo độ trễ thấp, phản hồi tức thì và thao tác mượt mà.
- **Đa phương thức (Multimodal Support)**: Hệ thống có khả năng tiếp nhận và xử lý đồng thời dữ liệu đầu vào dạng văn bản (Text) và hình ảnh (Image).
- **Tính thẩm mỹ Tiếp cận**: Giao diện hiện đại, hỗ trợ chế độ Dark/Light mode giúp giảm mỏi mắt và tăng tính cá nhân hóa.

1.2 Công nghệ sử dụng (Technology Stack)

Để đảm bảo hiệu năng cao và khả năng tùy biến sâu, nhóm phát triển đã lựa chọn kiến trúc Vanilla JavaScript (JS thuần), hạn chế phụ thuộc vào các Framework nặng nề.

- Ngôn ngữ cốt lõi:
  - HTML5: Xây dựng cấu trúc ngữ nghĩa (Semantic Web), đảm bảo tính tiếp cận (Accessibility).
  - CSS3:
    - Sử dụng CSS Variables (:root) để quản lý và chuyển đổi Theme động.
    - Sử dụng Flexbox Grid Layout để đảm bảo tính thích ứng (Responsive) trên đa thiết bị.
  - JavaScript (ES6+): Xử lý logic phía máy khách, quản lý trạng thái (State Management) và gọi API bất đồng bộ.
- UI/UX Design:
  - CSS Variables (:root) để quản lý Theme (Sáng/Tối) linh hoạt.
  - Flexbox Grid Layout cho bố cục responsive.
  - Font Inter từ Google Fonts giúp văn bản hiện đại, dễ đọc.
- Thư viện hỗ trợ:
  - Bộ phân tích cú pháp (Parser) giúp chuyển đổi phản hồi dạng Markdown từ AI sang HTML hiển thị đẹp mắt (hỗ trợ in đậm, bảng biểu, khối code)

- Font Inter (Google Fonts): Font chữ sans-serif hiện đại, tối ưu cho việc đọc trên màn hình kỹ thuật số.
- Swiper: (Mô đun mở rộng) Hỗ trợ hiển thị slider ảnh mượt mà cho các kết quả tìm kiếm.

1.3 Phân tích kiến trúc FRONTEND

1.3.1 File index.html - Bộ khung cấu trúc

Đóng vai trò xương sống của ứng dụng, chia thành các khu vực chức năng rõ ràng:

- Header: Chứa bộ điều hướng và công cụ tiện ích (Theme Toggle, Clear History).
- Main (chatBox): Vùng hiển thị luồng hội thoại động, nơi các phần tử DOM được JS thêm vào theo thời gian thực.
- Footer (Input Area): Khu vực nhập liệu phức hợp, bao gồm:
  - <textarea>: Thay thế thẻ input truyền thống để hỗ trợ nhập văn bản nhiều dòng và tự động co giãn chiều cao.
  - imagePreview: Vùng đệm hiển thị thumbnail ảnh trước khi upload.
- Components ẩn: Các thành phần như Lightbox (xem ảnh phóng to) được tải sẵn và chỉ hiển thị khi được kích hoạt.

1.3.2 File style.css - Định dạng Giao diện

Tập trung vào trải nghiệm thị giác (Visual Design):

- Cơ chế Theming: Khai báo toàn bộ bảng màu thông qua biến CSS (–bg-primary, –text-main). Khi đổi theme, chỉ cần thay đổi giá trị biến gốc, không cần viết lại CSS cho từng phần tử.
- Responsive Design: Thiết lập các Media Queries để đảm bảo giao diện không bị vỡ trên mobile (tự động ẩn bớt padding, thu nhỏ kích thước bong bóng chat).
- Micro-interactions: Các hiệu ứng nhỏ như hover, transition màu sắc, và animation khi tin nhắn mới xuất hiện.

1.3.3 File script.js - Logic xử lý trung tâm

Đây là thành phần quan trọng nhất, chịu trách nhiệm điều phối luồng dữ liệu:

- Quản lý trạng thái (State Management) với localStorage:
  - Lưu trữ lịch sử chat (chatHistory) và trạng thái giao diện (theme).
  - Giúp duy trì phiên làm việc liên tục ngay cả khi người dùng tải lại trang (F5).
- Xử lý hình ảnh Client-side:
  - Sử dụng FileReader để đọc file ảnh và chuyển đổi sang chuỗi Base64 (DataURL).
  - Ưu điểm: Cho phép hiển thị ảnh ngay lập tức (Instant Preview) mà chưa cần upload lên server, đồng thời an toàn khi lưu trữ cục bộ.

- Cơ chế giao tiếp bất đồng bộ (Asynchronous Communication):
  - Sử dụng fetch API kết hợp với async/await để gửi request mà không làm "đơ"giao diện người dùng.
  - Sử dụng FormData để đóng gói dữ liệu phức hợp (Multipart/form-data) gồm cả text và file binary.

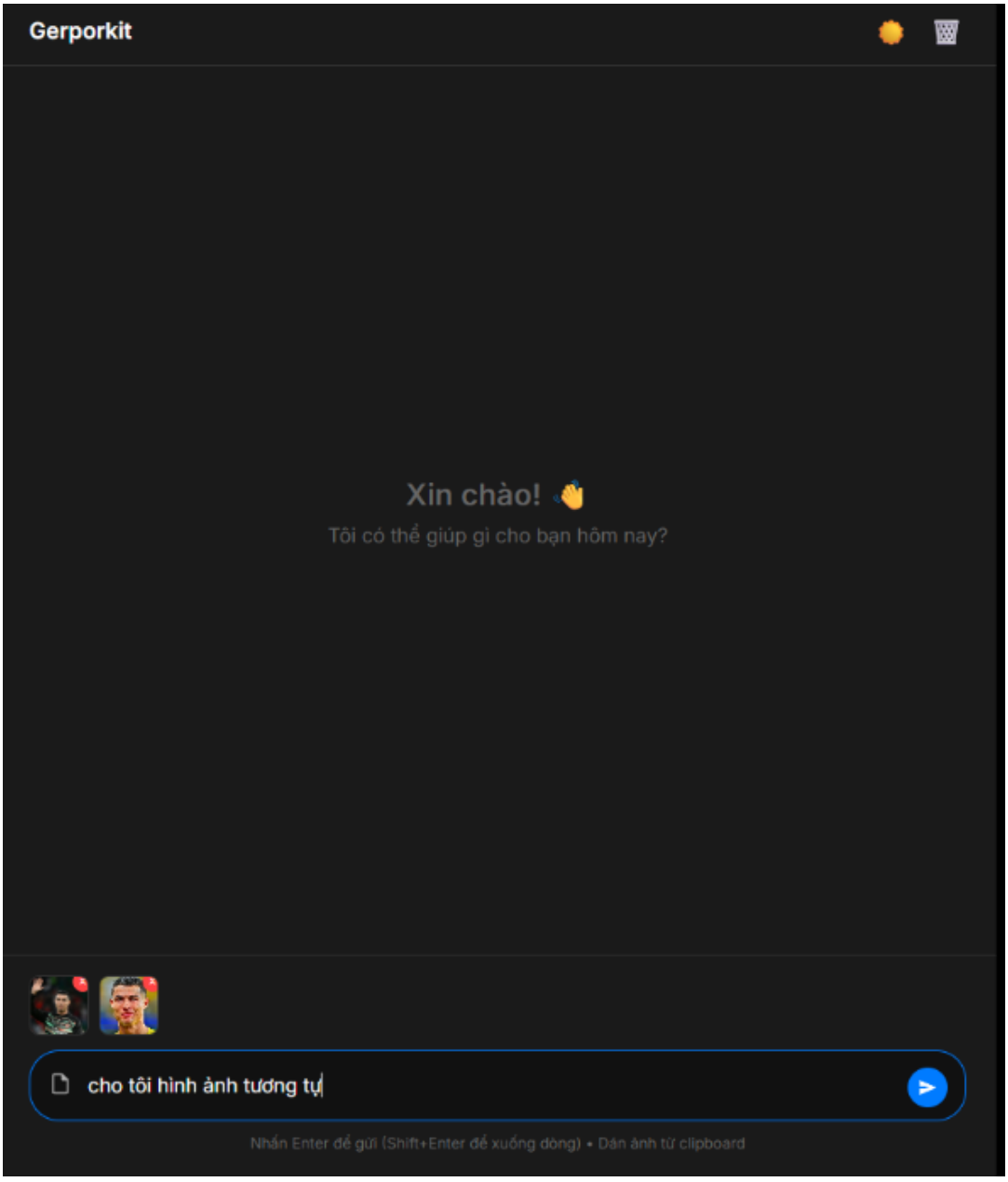
1.4 Các chức năng của hệ thống (System Features)

Hệ thống Frontend cung cấp các nhóm chức năng chính giúp người dùng tương tác thuận tiện với mô hình AI:

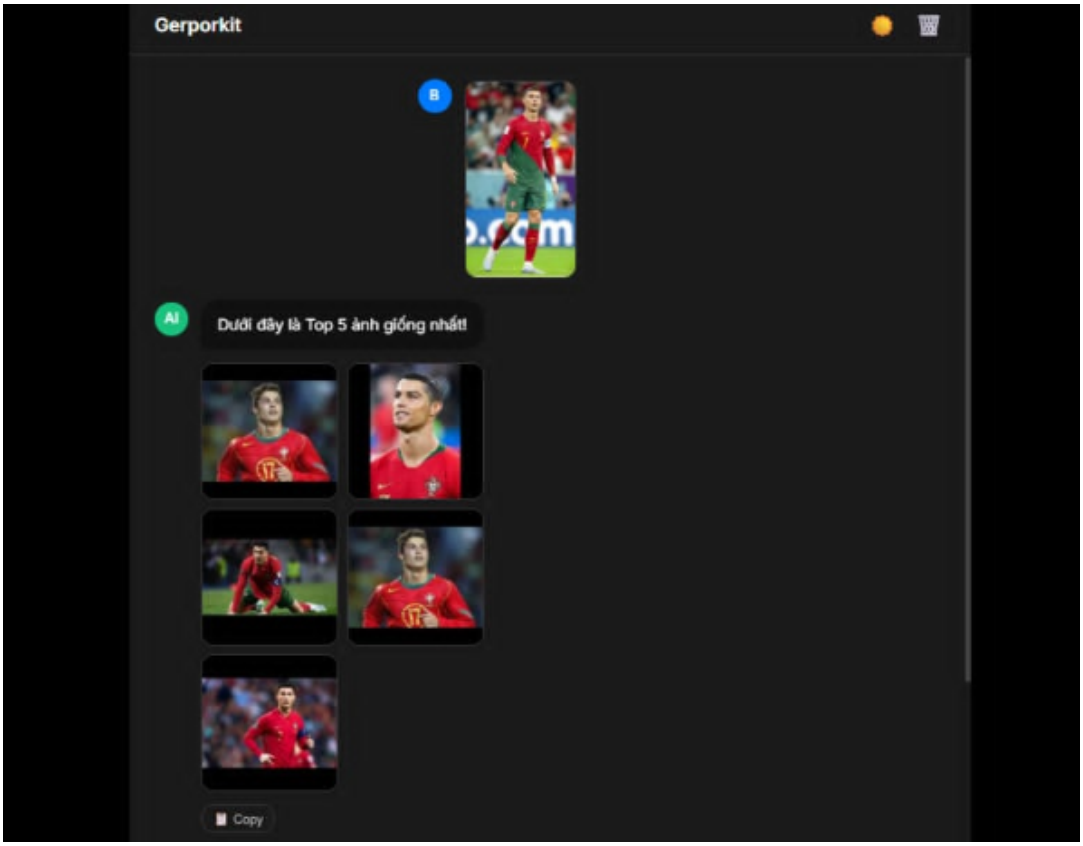
1.4.1 Chức năng Tương tác Tìm kiếm (Core Interaction)

Đây là nhóm chức năng chính giúp người dùng giao tiếp với AI.

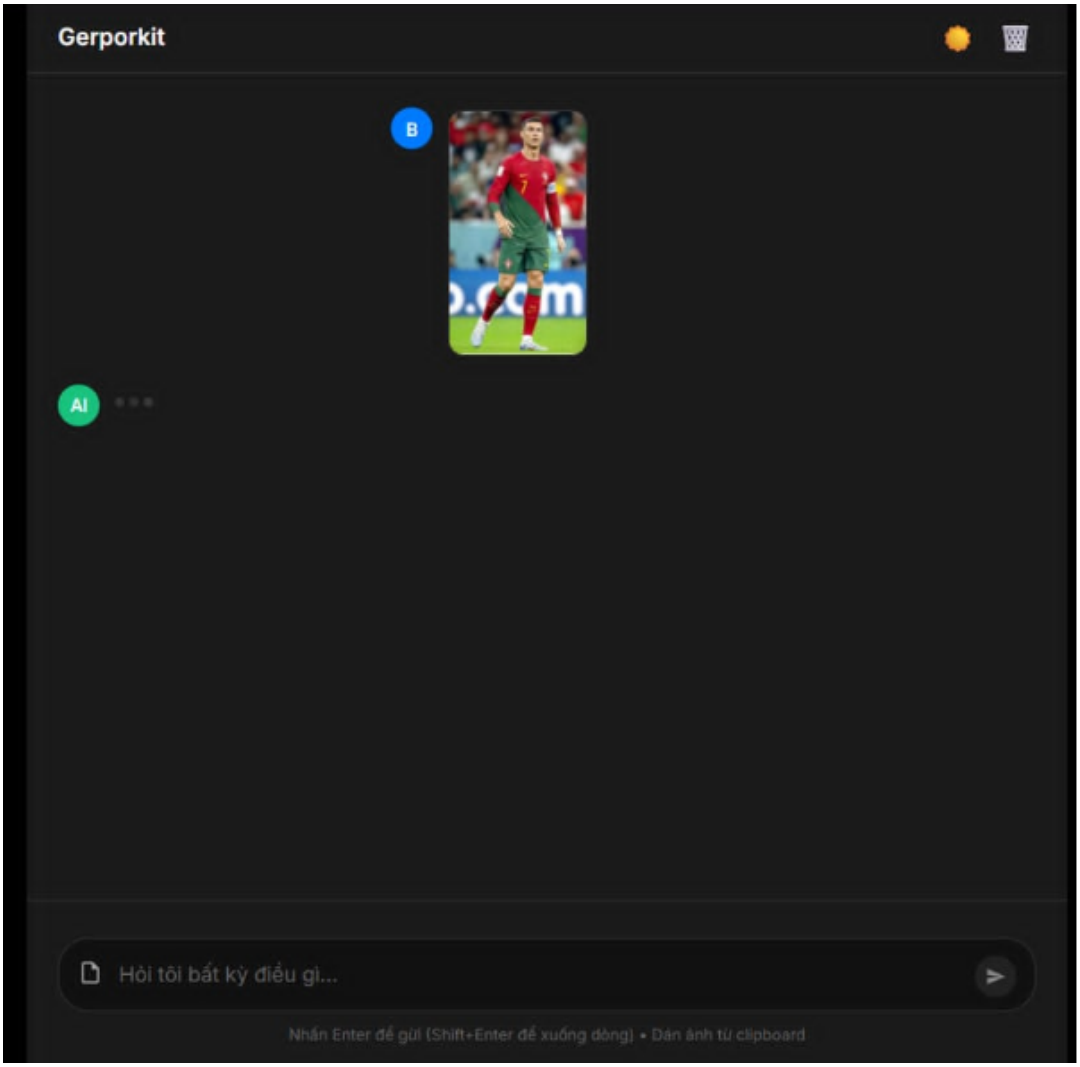
- **Gửi yêu cầu đa phương thức (Multimodal Query)**: Người dùng có thể gửi yêu cầu bằng văn bản (Text), hình ảnh (Image), hoặc kết hợp cả hai cùng lúc để tăng độ chính xác cho AI.



- **Hiển thị phản hồi thông minh**:
  - **Văn bản**: Kết quả từ AI được định dạng tự động (in đậm, in nghiêng, danh sách) giúp dễ đọc hơn văn bản thô.
  - **Hình ảnh minh họa**: Hệ thống hiển thị lưới ảnh (Grid view) do AI đề xuất kèm theo câu trả lời.



- **Trạng thái phản hồi (Feedback System)**: Hiển thị hiệu ứng "Đang nhập..."(Typing indicator) khi AI đang xử lý, giúp người dùng biết hệ thống vẫn đang hoạt động.



1.4.2 Chức năng quản lý đầu vào (Input Management)

Nhóm chức năng hỗ trợ người dùng thao tác dữ liệu trước khi gửi đi:

- **Dán ảnh nhanh (Clipboard Paste)**: Hỗ trợ dán ảnh trực tiếp từ bộ nhớ tạm (Ctrl+V) vào khung chat mà không cần lưu file về máy.
- **Xem trước ảnh (Image Preview)**: Hiển thị ảnh thu nhỏ ngay khi người dùng chọn file hoặc dán ảnh.
- **Quản lý file**: Cho phép xóa bỏ từng ảnh trong danh sách chờ trước khi nhấn nút Gửi.

```
// DÁN ẢNH TỪ CLIPBOARD
document.addEventListener("paste", async (e) => {
  const items = e.clipboardData?.items;
  if (!items) return;

  const filePromises = [];
  for (const item of items) {
    if (item.kind === "file" && item.type.startsWith("image/")) {
      const file = item.getAsFile();
      if (file) {
        // SỬA LỖI ẢNH: Chuyển file thành dataURL
        filePromises.push(readFileAsDataURL(file));
      }
    }
  }

  if (filePromises.length > 0) {
    e.preventDefault();
    const newImages = await Promise.all(filePromises);
    images = images.concat(newImages).slice(0, 5); // Lưu object {file, dataURL}
    updatePreview();
  }
});
```

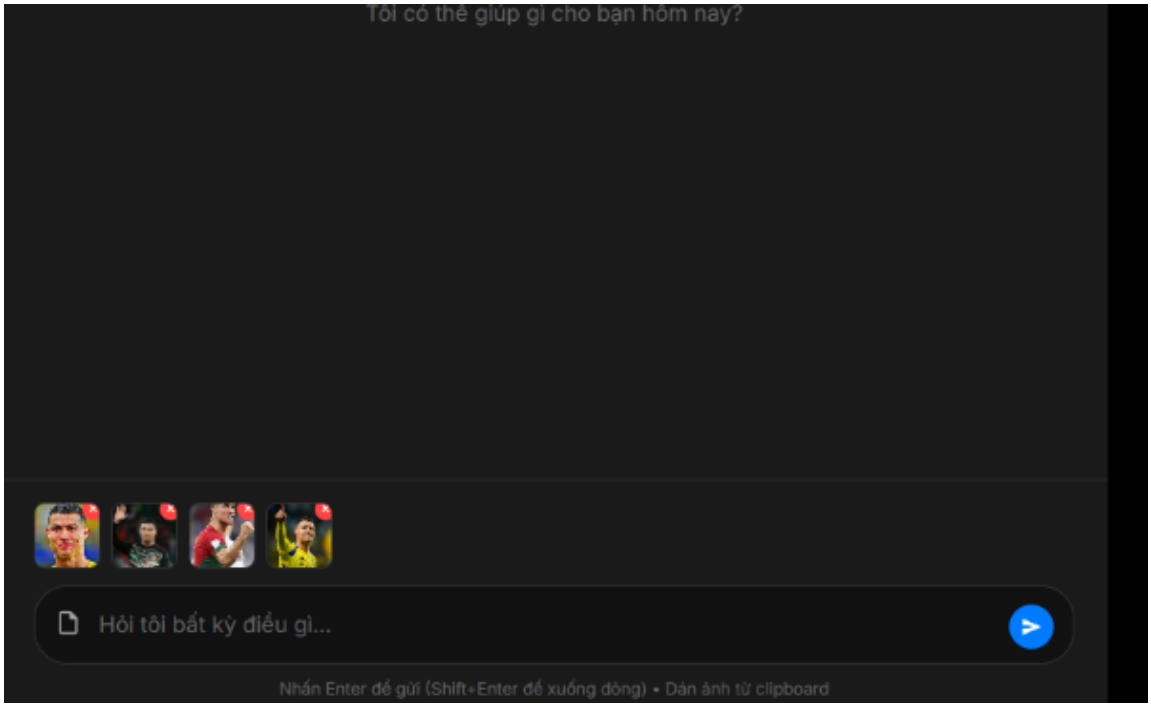
Dán ảnh nhanh bằng Ctrl+V

```
// UPLOAD ẢNH
imageUpload.addEventListener("change", async () => {
  const files = Array.from(imageUpload.files);
  // SỬA LỖI ẢNH: Chuyển các file thành dataURL
  const filePromises = files.map(readFileAsDataURL);

  const newImages = await Promise.all(filePromises);
  images = images.concat(newImages).slice(0, 5); // Lưu object {file, dataURL}
  updatePreview();
  imageUpload.value = "";
});

// XÓA ẢNH
imagePreview.addEventListener("click", (e) => {
  if (e.target.tagName === "BUTTON") {
    const idx = parseInt(e.target.dataset.idx);
    images.splice(idx, 1);
    updatePreview();
  }
});
```

Upload ảnh từ nút đính kèm và xóa ảnh

```
function updatePreview() {
  imagePreview.innerHTML = "";
  // SỬA LỖI ẢNH: Dùng dataURL để hiển thị preview
  images.forEach((imgData, i) => {
    const div = document.createElement("div");
    div.className = "preview-item";
    div.innerHTML = `
      <img src="${imgData.dataURL}" alt="preview" />
      <button data-idx="${i}">x</button>
    `;
    imagePreview.appendChild(div);
  });
}
```

Hiển thị preview ảnh thu nhỏ



Hình ảnh minh họa

1.4.3 Chức năng tiện ích cá nhân hóa (Utilities UI/UX)

Các tính năng bổ trợ giúp nâng cao trải nghiệm sử dụng:

- **Chế độ giao diện (Dark/Light Mode)**: Cho phép người dùng chuyển đổi qua lại giữa giao diện Sáng và Tối để phù hợp với điều kiện ánh sáng môi trường.

```javascript
// ===== TÍNH NĂNG MỚI: THEME TOGGLE =====
if (themeToggle) {
  // Load saved theme
  const savedTheme = localStorage.getItem('theme') || 'dark';
  document.body.dataset.theme = savedTheme;

  themeToggle.addEventListener('click', () => {
    const current = document.body.dataset.theme;
    const newTheme = current === 'dark' ? 'light' : 'dark';
    document.body.dataset.theme = newTheme;
    localStorage.setItem('theme', newTheme);
  });
}
```

```css
:root {
  /* Biến màu cho theme tối (mặc định) */
  --bg-primary: #000;
  --bg-secondary: #1a1a1a;
  --bg-tertiary: #111;
  --text-primary: #fff;
  --text-secondary: #aaa;
  --text-muted: #666;
  --border-color: #333;
  --accent-color: #007bff;
  --accent-hover: #0056b3;
  --danger-color: #ff4444;
}

body[data-theme="light"] {
  /* Biến màu cho theme sáng */
  --bg-primary: #f0f2f5;
  --bg-secondary: #ffffff;
  --bg-tertiary: #f9f9f9;
  --text-primary: #111;
  --text-secondary: #555;
  --text-muted: #888;
  --border-color: #ddd;
  --accent-color: #007bff;
  --accent-hover: #0056b3;
  --danger-color: #ff4444;
}
```

- **Lưu trữ lịch sử (Chat History)**: Tự động lưu lại toàn bộ cuộc hội thoại vào trình duyệt. Người dùng có thể tải lại trang (F5) mà không mất nội dung cũ.

```javascript
// Load lịch sử (Giờ đây đã an toàn vì ảnh là dataURL)
if (chatHistory.length === 0) {
  showEmptyState();
} else {
  chatHistory.forEach(msg => addMessage(msg.sender, msg.text, msg.images || [], false, msg.isMarkdown));
}
```

- **Xem ảnh chi tiết (Lightbox)**: Khi nhấm vào ảnh nhỏ trong đoạn chat, ảnh sẽ được phóng to toàn màn hình để xem rõ chi tiết.

```javascript
// ===== TÍNH NĂNG MỚI: LIGHTBOX ẢNH =====
function openLightbox(url) {
  const lightbox = document.getElementById('lightbox');
  const lightboxImg = document.getElementById('lightboxImg');
  if (lightbox && lightboxImg) {
    lightboxImg.src = url;
    lightbox.classList.add('active');
  }
}

// Setup lightbox events
const lightbox = document.getElementById('lightbox');
const lightboxClose = document.getElementById('lightboxClose');
if (lightbox && lightboxClose) {
  lightboxClose.onclick = () => lightbox.classList.remove('active');
  lightbox.onclick = (e) => {
    if (e.target === lightbox) lightbox.classList.remove('active');
  };
}
```

```css
/* === Lightbox (mới) === */
.lightbox {
  display: none;
  position: fixed;
  z-index: 999;
  padding-top: 60px;
  left: 0; top: 0;
  width: 100%; height: 100%;
  overflow: auto;
  background-color: rgba(0,0,0,0.9);
}
.lightbox.active { display: block; }
.lightbox-content {
  margin: auto;
  display: block;
  max-width: 90%;
  max-height: 90vh;
}
.lightbox-close {
  position: absolute;
  top: 15px; right: 35px;
  color: #f1f1f1;
  font-size: 40px;
  font-weight: bold;
  cursor: pointer;
}
```

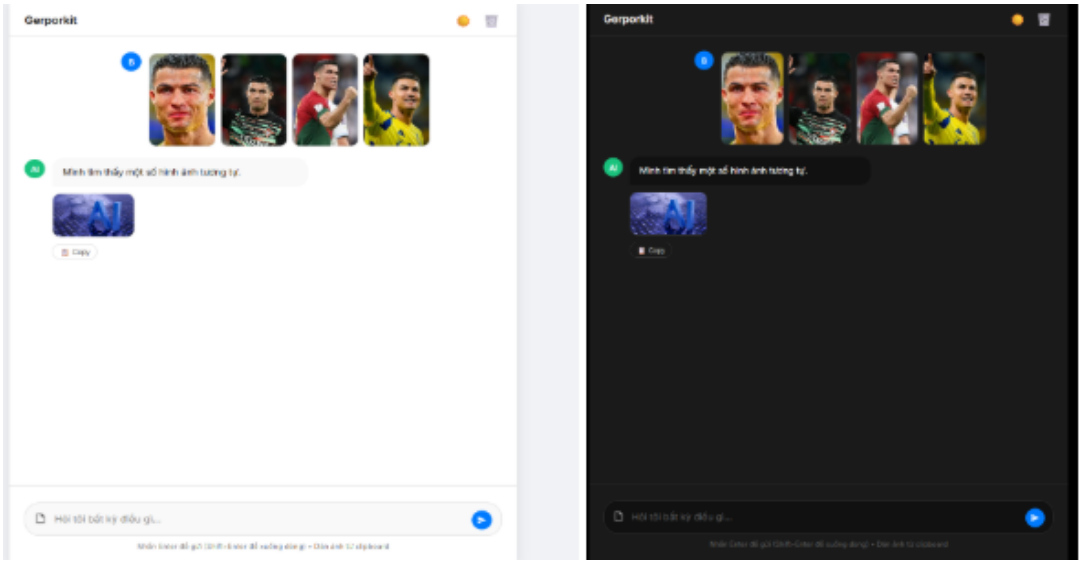
- **Sao chép nhanh (Quick Copy)**: Cung cấp nút sao chép nội dung câu trả lời của AI chỉ với 1 cú click.

```
// ===== TÍNH NĂNG MỚI: NÚT COPY CHO BOT =====
if (sender === "bot" && text && !isTyping) {
  const actions = document.createElement("div");
  actions.className = "message-actions";
  actions.innerHTML = `<button class="action-btn copy-btn">📄 Copy</button>`;
  actions.querySelector('.copy-btn').onclick = () => copyText(text);
  content.appendChild(actions);
}
```

- **Xóa lịch sử**: Chức năng cho phép dọn dẹp toàn bộ dữ liệu chat để bắt đầu phiên làm việc mới.

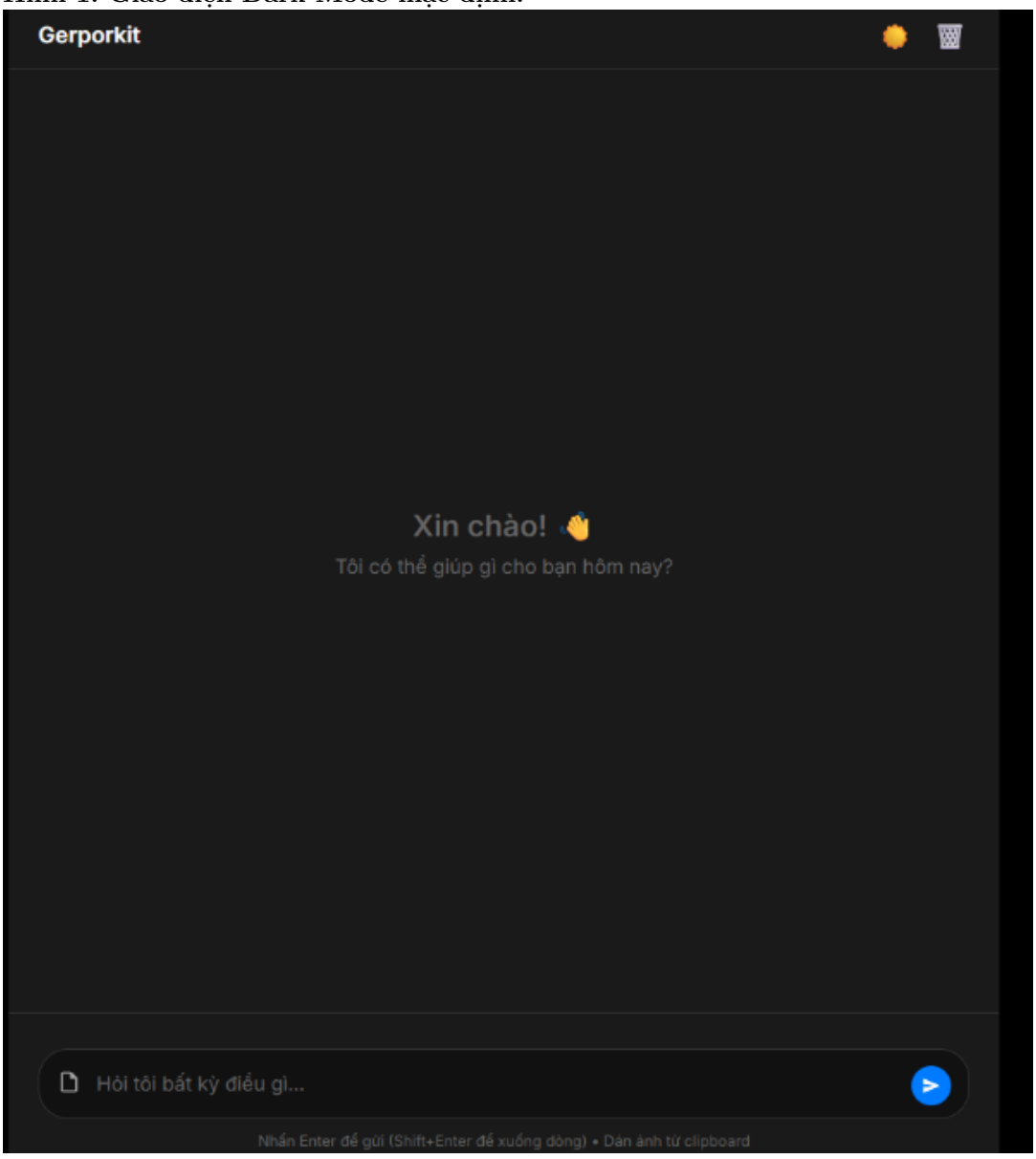
```
// ===== TÍNH NĂNG MỚI: XÓA LỊCH SỬ =====
if (clearBtn) {
  clearBtn.addEventListener('click', () => {
    if (confirm('Bạn có chắc muốn xóa toàn bộ lịch sử chat?')) {
      chatHistory = [];
      localStorage.removeItem('chatHistory');
      chatBox.innerHTML = '';
      showEmptyState();
    }
  });
}
```

- **Minh họa**

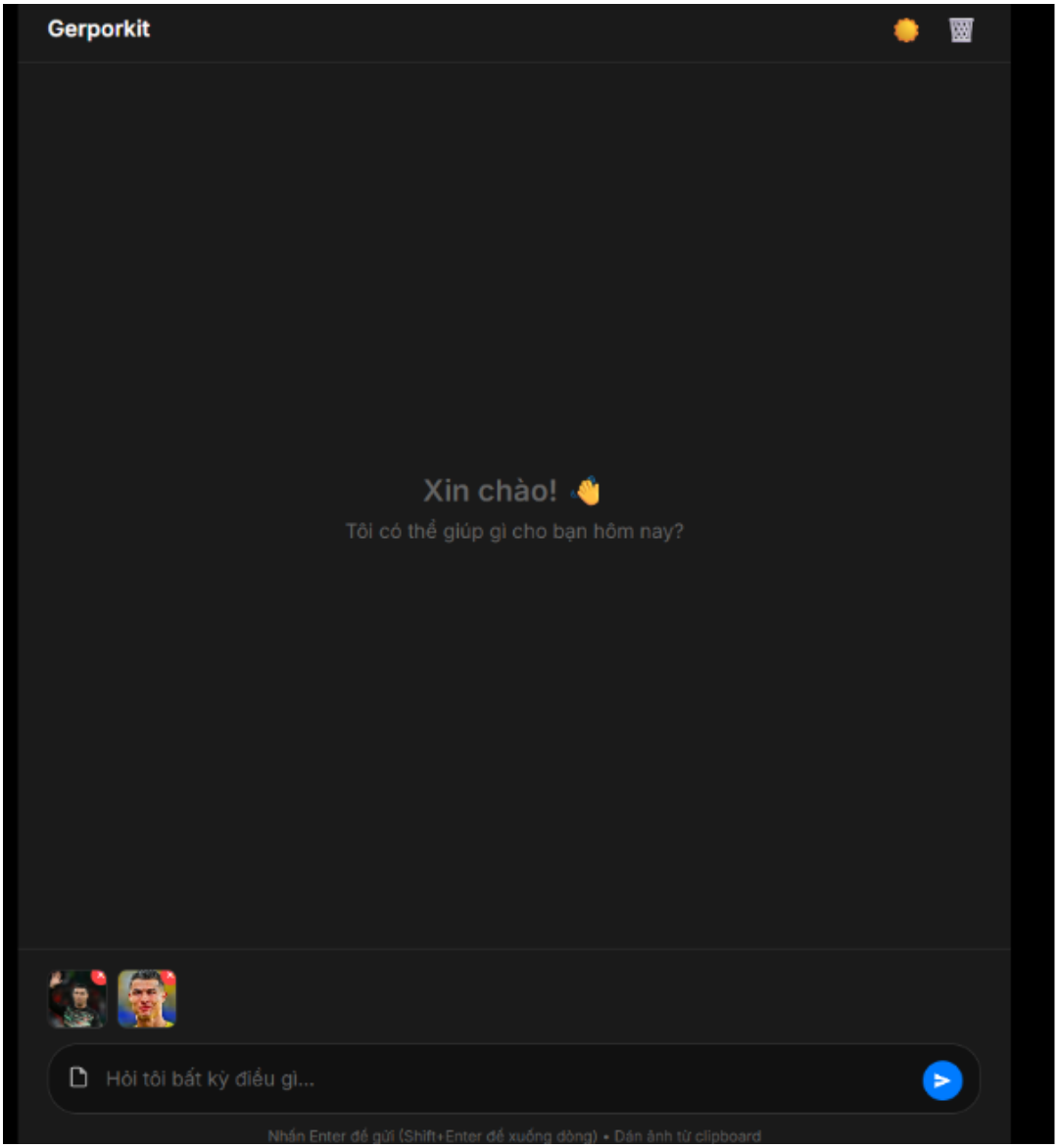

Gerporkit
Gerporkit

1.4.4 Giao diện và Thiết kế (UI Showcase)

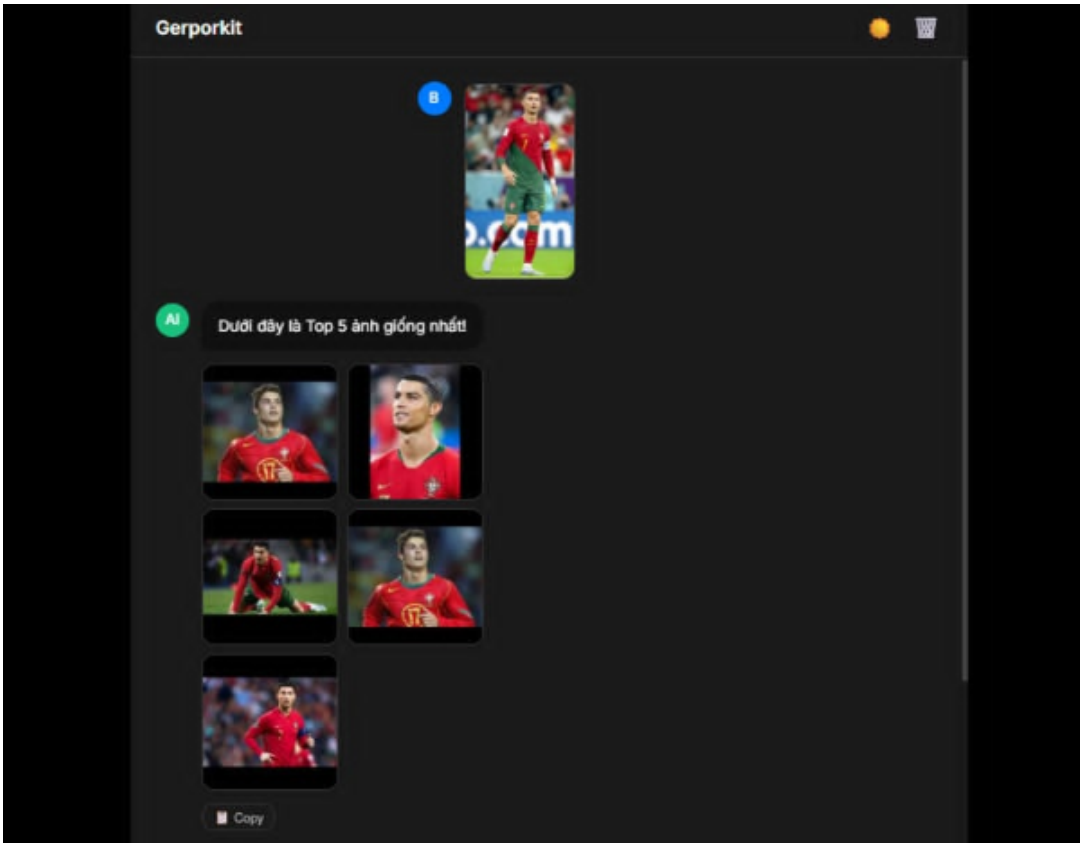
- **Hình 1: Giao diện Dark Mode mặc định.**



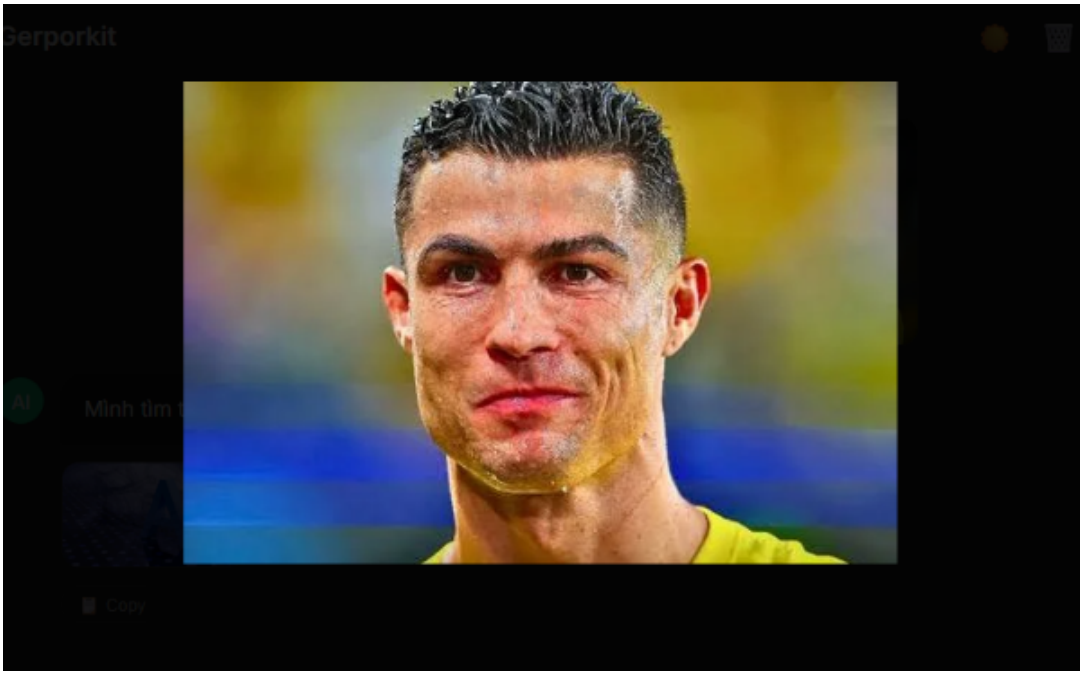
- **Hình 2: Tính năng Preview ảnh và Paste từ Clipboard.**



- **Hình 3: Hiển thị kết quả Markdown.**



- **Hình 4: Lightbox phóng to ảnh**



1.4.5 Kết luận Hướng phát triển

Phần Frontend đã hoàn thành tốt vai trò tiếp nhận và hiển thị thông tin cho mô hình AI. Hệ thống đảm bảo tính tương tác cao (Interactive), tốc độ phản hồi nhanh nhờ xử lý Client-side (Preview ảnh, Markdown render), và code sạch, dễ bảo trì.
Có thể nâng cấp thêm:

- Thêm tính năng Voice-to-Text (Nhập liệu bằng giọng nói).
- Streaming Response (Hiển thị chữ chạy từng từ giống ChatGPT thay vì chờ toàn bộ câu).

2 QUY TRÌNH XÂY DỰNG HỆ THỐNG API GOOGLE COLAB (MODEL AI) - BACKEND - FRONTEND (IMAGE)

Frontend → FastAPI Backend → Colab AI Model → Dataset → Colab Response → Backend → Frontend

2.1 Giới thiệu chung

Đề tài xây dựng một hệ thống AI Search cho phép người dùng tải ảnh lên và tìm các ảnh tương tự trong dataset nhờ mô hình trích xuất đặc trưng CLIP.
Hệ thống gồm ba phần chính:

- **Xây dựng một API** chạy trên Google Colab để xử lý ảnh và trả về kết quả tìm kiếm

- **Backend FastAPI/Flask API** (nhận ảnh → gửi sang Colab → xử lý → trả kết quả)
- **Frontend Web UI** (giao diện chat AI, upload ảnh, nhận kết quả base64)

2.2 Thiết kế API trên Google Colab

2.2.1 Mục đích

- Colab chạy mô hình AI (CLIP).
- dữ liệu đầu vào → xử lý → trả về top 5 ảnh tương tự dưới dạng base64.

2.2.2 Các bước thực hiện

1. Cài đặt Môi trường (Environment Setup)
   Đây là bước chuẩn bị nguyên vật liệu. Colab mặc định chưa có các thư viện để chạy web server, nên ta phải cài thêm.

   - fastapi & uvicorn: Bộ đôi Framework và Server để tạo API siêu tốc.
   - pyngrok: Công cụ tạo "đường hầm"(Tunnel) để public cổng 8000 của Colab ra Internet (vì Colab chạy trên máy ảo kín, bên ngoài không gọi vào được nếu thiếu cái này).
   - python-multipart: Thư viện quan trọng để xử lý việc upload file ảnh (dạng multipart/form-data).
   - deep-translator: "Người phiên dịch"giúp hệ thống hiểu được tiếng Việt của người dùng.

2. Xử lý Logic API (The Controller)
   Đây là cell quan trọng nhất, nơi định nghĩa logic "tiếp khách"của Server.

   - Khởi tạo Cấu hình Server (Server Setup)
     - Tạo App: Khởi tạo ứng dụng FastAPI.
     - Cấu hình CORS: Mở quyền truy cập (allow_origins=["*"]) để đảm bảo Backend Local (hoặc bất kỳ ai) cũng có thể gọi vào API này mà không bị chặn bởi trình duyệt/network.
     - Chuẩn bị công cụ: Khởi tạo GoogleTranslator để sẵn sàng dịch thuật ngữ nghĩa.

```
app = FastAPI()

# Câú hình CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

# Khởi tạo dịch thuật
translator = GoogleTranslator(source='auto', target='en')
```

- Xử lý & Chuẩn hóa Đầu vào (Input Processing)
  Đây là khâu tiền xử lý quan trọng để biến dữ liệu thô thành dữ liệu máy hiểu được:
  - Với Văn bản (Text): Nhận chuỗi tiếng Việt → Dịch sang tiếng Anh (ví dụ: "con mèo> "cat") → Gọi hàm extract_text_features để biến thành vector.
  - Với Hình ảnh (Image): Nhận file upload (dạng bytes) → Chuyển thành đối tượng ảnh (PIL Image) → Gọi hàm extract_image_features để biến thành vector. Cell này cũng xử lý logic để tương thích với hàm extract cũ (chuyển đổi Tensor sang Numpy nếu cần).

```
try:
    # 1. Xư'lý Text (Dịch + Extract bằng hàm có săñ của bạn)
    if query:
        translated = translator.translate(query)
        print(f"   Dịch: {translated}")
        # Gọi hàm extract_text_features đã có trong notebook
        text_vec = extract_text_features(translated)

    # 2. Xư'lý Ảnh (Extract bằng hàm có săñ của bạn)
    if file:
        content = await file.read()
        pil_img = Image.open(io.BytesIO(content)).convert("RGB")
        # Gọi hàm extract_image_features đã có trong notebook
        image_vec = extract_image_features(pil_img)
```

- Thuật toán Tìm kiếm Kết hợp (Hybrid Logic)
  Đây là "bộ não"thông minh của hệ thống:
  - Sử dụng hàm combine_query để trộn vector văn bản và vector hình ảnh lại với nhau dựa trên trọng số alpha.
  - Điều này cho phép tìm kiếm linh hoạt: chỉ bằng ảnh, chỉ bằng chữ, hoặc cả hai (ví dụ: ảnh cái áo + chữ "màu xanh").

```
# 3. Kết hợp (Dùng hàm combine_query có sẵn)
if text_vec is not None or image_vec is not None:
    final_vec = combine_query(torch.tensor(text_vec) if text_vec is not None else None,
                              torch.tensor(image_vec) if image_vec is not None else None,
                              alpha)

    if isinstance(final_vec, torch.Tensor):
        final_vec = final_vec.cpu().numpy()

    final_vec = final_vec.reshape(1, -1)
```

- Truy xuất & Xếp hạng (Retrieval & Ranking)
  - Thực hiện phép tính toán học Cosine Similarity giữa vector truy vấn (vừa tạo ở bước 3) với hàng nghìn vector trong cơ sở dữ liệu (vectors_db).
  - Sắp xếp điểm số từ cao xuống thấp và lấy ra Top 5 kết quả giống nhất.

```
# 4. Tính toán Similarity
scores = []
for path, vec in vectors_db.items():
    vec = vec.reshape(1, -1)
    score = cosine_similarity(final_vec, vec) # Hàm có sẵn
    scores.append((path, float(score)))

scores.sort(key=lambda x: x[1], reverse=True)

results = []
for path, score in scores[:5]:
    b64 = image_to_base64(path)
    if b64: results.append({"image": b64, "score": score})

return {"top5": results}
```

- Đóng gói Kết quả (Response Formatting)
  - Thay vì chỉ trả về đường dẫn file (trên Colab vốn không truy cập được từ bên ngoài), code sử dụng hàm image_to_base64.
  - Nó đọc file ảnh kết quả từ ổ đĩa Colab → Mã hóa thành chuỗi ký tự Base64 → Gửi trọn gói về cho Backend Local. Nhờ vậy, Frontend có thể hiển thị ảnh ngay lập tức.

```
def image_to_base64(path):
    try:
        if not os.path.exists(path): return None
        with open(path, "rb") as img_file:
            return "" + base64.b64encode(img_file.read()).decode('utf-8')
    except:
        return None
```

2.2.3 Chạy server

```python
# Chạy Server và Public qua Ngrok
import uvicorn
import nest_asyncio
from pyngrok import ngrok

# --- ĐIỀN TOKEN CỦA BẠN VÀO ĐÂY ---
NGROK_TOKEN = "35txrgcicOrnCwU6m0F6zWoDvm6_3uaLFqCTzWNyqmnCgDCCx"
ngrok.set_auth_token(NGROK_TOKEN)

# Mở tunnel port 8000
ngrok_tunnel = ngrok.connect(8000)
print('🚀 PUBLIC URL CỦA BẠN:', ngrok_tunnel.public_url)
print('👉 Hãy copy URL này dán vào file config.py ở backend!')

# Chạy server
nest_asyncio.apply()

# Cấu hình server thủ công
config = uvicorn.Config(app, host="0.0.0.0", port=8000)
server = uvicorn.Server(config)

# Chạy server bằng await (tương thích với Colab)
await server.serve()
```

2.2.4 Quy trình

1. Cell 1: Trang bị Công cụ (Installation)
   - Hành động: Cài đặt các thư viện Python cần thiết mà môi trường Colab mặc định chưa có.
   - Thành phần chính: fastapi (tạo web app), uvicorn (chạy server), pyngrok (mở cổng ra internet), và deep-translator (dịch tiếng Việt).
2. Cell 2: Kết nối Tài nguyên (Resource Linking)
   - Quy trình:
     - Kiểm tra xem bạn đã chạy các cell huấn luyện bên trên chưa.
     - Gán biến vectors_db trỏ vào biến loaded (dữ liệu vector) có sẵn.
     - Kết nối với model và processor đang nằm trên GPU.
3. Cell 3: Thiết lập Logic Xử lý (API Controller)
   Đây là "bộ não"điều khiển luồng dữ liệu của Server.
   - Khởi tạo App: Tạo ứng dụng FastAPI và cấu hình CORS để cho phép kết nối từ bên ngoài.

- Xử lý Đầu vào (Input):
  - Nếu là Text: Dùng GoogleTranslator dịch từ Việt sang Anh -> gọi extract_text_features.
  - Nếu là Ảnh: Đọc file upload -> gọi extract_image_features (đã chỉnh sửa để nhận object ảnh thay vì đường dẫn file).
- Xử lý Tìm kiếm (Core Logic):
  - Dùng hàm combine_query để trộn vector ảnh và vector chữ (Hybrid Search).
  - Tính toán cosine_similarity với toàn bộ Database.
  - Sắp xếp và lấy ra Top 5 kết quả tốt nhất.
- Xử lý Đầu ra (Output): Mã hóa ảnh kết quả thành chuỗi Base64 để gửi trả về cho Backend/Frontend hiển thị ngay lập tức.

4. Cell 4: Kích hoạt Công khai (Deployment)
   - Hành động: Khởi động Server và mở đường truyền.
   - Quy trình:
     - Đăng nhập Ngrok bằng Token của bạn.
     - Tạo đường hầm (Tunnel) từ cổng 8000 của Colab ra địa chỉ Public Internet.
     - Sử dụng cấu hình uvicorn.Config và await server.serve() để chạy server ở chế độ bất đồng bộ, khắc phục lỗi xung đột asyncio trên môi trường Notebook.

   Tóm lại: Quy trình này đi từ việc Cài đặt -> Tận dụng tài nguyên cũ -> Định nghĩa cách xử lý -> Mở cổng kết nối.

2.3 Thiết kế Backend FastAPI

2.3.1 Mục đích

- Làm cầu nối giữa frontend và Colab API.
- Bảo vệ frontend khỏi lỗi CORS hoặc lỗi mạng từ Colab.
- Chuẩn hóa dữ liệu trước khi trả về.

2.3.2 Cấu trúc backend

routes/image_route.py:

- Mỗi route là một API endpoint mà frontend có thể gọi.
- Chỉ xử lý nhiệm vụ nhận request và đưa request vào service.

services/image_service.py: Xử lý logic chính của hệ thống:

- đọc dữ liệu đầu vào
- chuẩn hóa dữ liệu
- gửi ảnh sang API Colab

- nhận kết quả từ AI
- encode lại dữ liệu đúng format frontend

utils/config.py: chứa API colab main.py:

- Tạo ứng dụng API chính.
- Cho phép mọi domain gọi API (tránh lỗi CORS khi frontend gọi backend).

2.3.3 Luồng xử lý dữ liệu Backend

- Frontend → (1) gửi FormData (query, file)
- Route → (2) nhận request, chuẩn hóa input
- Service → (3) gửi file sang Colab API
- Colab API → (4) xử lý AI, trả top 5 ảnh base64
- Service → (5) convert + format JSON
- Route → (6) trả JSON về frontend
- Frontend → (7) hiển thị ảnh và text

2.4 Kết nối Backend Frontend

2.4.1 Bật CORS trong Backend

Cho phép mọi domain (localhost hoặc deploy) gọi API backend

```
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"]
)
```

2.4.2 Frontend gọi API backend hiện tại

```
const res = await fetch("http://127.0.0.1:8000/api/predict-image", {
  method: "POST",
  body: fd
});
```

2.4.3 Chuẩn hóa input và ouput cho frontend và backend

- Frontend và backend thống nhất về định dạng input, output
- Frontend gửi input đúng định dạng mà Backend cần(dạng UploadFile): `fd.append("file", imgData.file);`
- Backend: trả về output đúng format Frontend cần

2.5 Kiểm thử hệ thống

- **Test backend trên cổng http://127.0.0.1:8000/docs**
  - Gửi thử 1 ảnh → backend → Colab → backend → trả kết quả
  - API chạy đúng, trả về top 5 ảnh.
- **Test trực tiếp trên frontend**
  - Tải ảnh lên
  - Backend nhận ảnh
  - Colab xử lý
  - Frontend hiển thị hình ảnh tương tự → Hoàn tất luồng xử lý.

2.6 Kết luận

Hoàn thành các nhiệm vụ:

1. Xây dựng API AI tìm ảnh tương tự trên Google Colab.
2. Thiết kế backend FastAPI làm cầu nối, xử lý dữ liệu và trả kết quả chuẩn JSON.
3. Kết nối thành công frontend với backend và backend với Colab.
4. Hoàn thiện luồng gửi ảnh → AI xử lý → nhận kết quả và hiển thị lên giao diện

TÀI LIỆU THAM KHẢO

Một số tài liệu tham khảo chính:

1. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. Bài báo giới thiệu mô hình CLIP (Contrastive Language-Image Pre-training), nền tảng cốt lõi cho việc truy vấn ảnh bằng văn bản thông qua không gian vector chung. Truy cập tại: arxiv.org/abs/2103.00020

2. Dubreuil, A., & Galmar, B. (2022). *A Survey on Deep Learning for Content-Based Image Retrieval*. Bài tổng quan nghiên cứu các kỹ thuật học sâu hiện đại được áp dụng trong hệ thống truy vấn hình ảnh dựa trên nội dung (CBIR). Truy cập tại: arxiv.org/abs/2201.10344

3. Johnson, J., Douze, M., & Jégou, H. (2019). *Billion-scale similarity search with GPUs*. Nghiên cứu về thư viện Faiss và các thuật toán tối ưu hóa tìm kiếm độ tương đồng (Similarity Search) trên quy mô lớn bằng GPU. Truy cập tại: arxiv.org/abs/1702.08734

4. Chen, W., Liu, Y., Kirillov, A., & Darrell, T. (2023). *A Comprehensive Survey on Hybrid Visual Retrieval*. Tài liệu phân tích các phương pháp truy vấn kết hợp (Hybrid Retrieval) giữa văn bản và hình ảnh, tương tự như cơ chế `combine_query` trong mã nguồn. Truy cập tại: arxiv.org/abs/2304.14371

5. Shao, Z., Zhang, Z., & Lin, Z. (2021). *Deep Learning for Image Retrieval: A Survey*. Bài báo cung cấp cái nhìn toàn diện về quá trình tiến hóa của các phương pháp trích xuất đặc trưng từ truyền thống đến các kiến trúc Transformer hiện đại. Truy cập tại: arxiv.org/abs/2101.11282