RAYAN Challenge - Phase 2

Overview

This challenge focuses on developing an intelligent search system that understands both visual and textual information to identify relevant images from a database. Participants must create a system capable of processing queries that combine visual content with textual descriptions to find the most accurate matches.

Challenge Objective

The task requires building a system that can:

1. Process input consisting of:
    - A reference image containing a visual scene
    - A text description providing additional context or modifications
2. Identify and return the single most relevant matching image from a provided database

Figure 1 serves as an example of this task.

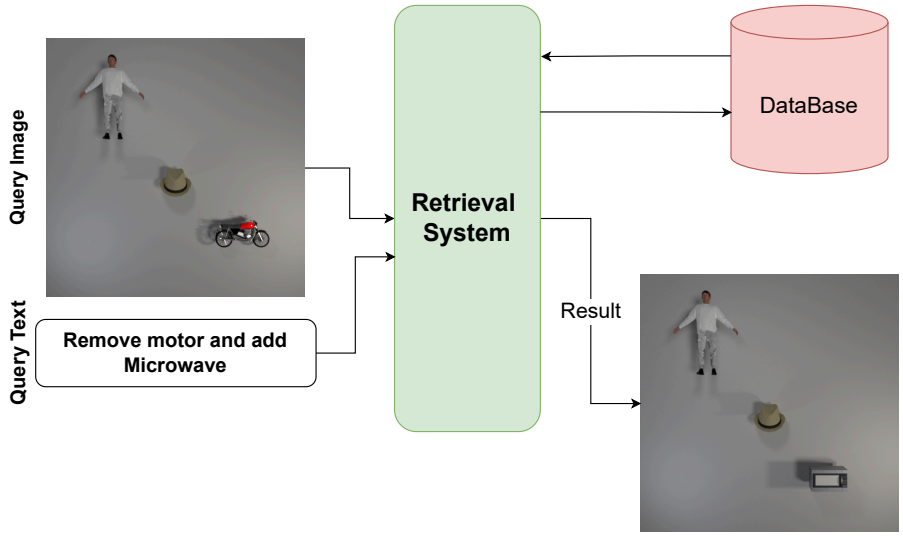


Figure 1: Example of query processing and result retrieval.

Technical Requirements

Participants are required to build a model that accepts an image and a text as input, processes the query, and retrieves the most relevant image from the database. The dataset provided contains pairs of image and text as queries, along with the expected target images for retrieval. This dataset is meant for training and fine-tuning your models.

We will evaluate your submission on a specially curated test set, which differs significantly from the provided training dataset. The test set will feature more diverse text descriptions and novel combinations of visual elements in the images. This is to assess the model's generalization capability. You can view some examples of these challenging test cases through the link provided at the end of this document.

It is crucial that your model does not overfit to the training set but instead generalizes well to unseen data. To this end, we encourage you to focus on developing a robust system capable of handling novel and diverse queries. Techniques like data augmentation, transfer learning, and careful fine-tuning can enhance the generalization of your model

Model Constraints

- Maximum model size: 4GB
- **Not Allowed in Final Model for Inference**:
  - Large Language Models (LLMs)
  - Object detection models
  - Pre-trained models that directly solve the task without modifications
- **Allowed**:
  - Pre-trained Vision-Language Models (e.g., from OpenCLIP) if fine-tuned for this task

Submission Requirements

Required Files

Your submission must include:

1. `model.py`: Implementation of your model
2. `generate_embeds.py`: Contains two essential functions:
   - `encode_queries(DataFrame)`
   - `encode_database(DataFrame)`
3. `weights.pth`: Trained model weights
4. `requirements.txt`: Environment dependencies

encode_queries Function

This function processes query pairs (image + text) and must follow this signature:

```python
def encode_queries(df: pd.DataFrame) -> np.ndarray:
    """
    Process query pairs and generate embeddings.

    Args:
        df (pd.DataFrame): DataFrame with columns:
            - query_image: str, paths to query images
            - query_text: str, text descriptions

    Returns:
        np.ndarray: Embeddings array (num_queries, embedding_dim)
    """
    pass
```

encode_database Function

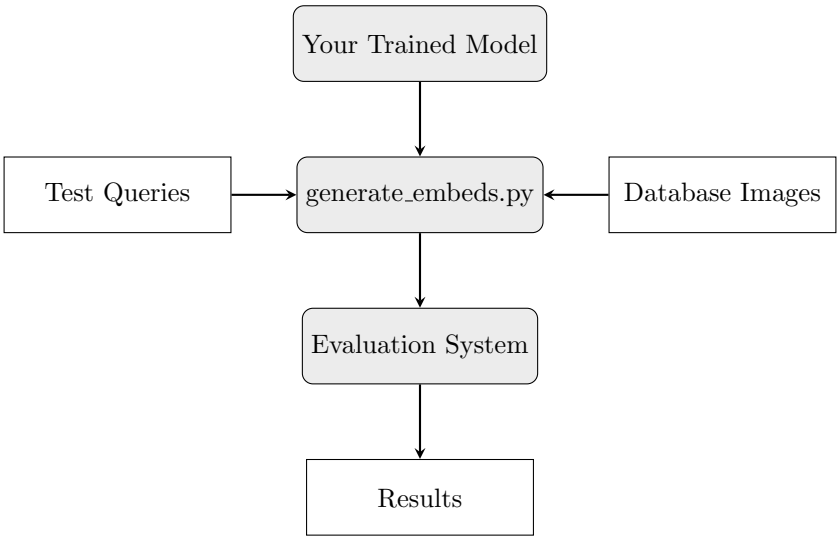
This function processes database images and must follow this signature:

```python
def encode_database(df: pd.DataFrame) -> np.ndarray:
    """
    Process database images and generate embeddings.

    Args:
        df (pd.DataFrame): DataFrame with column:
            - target_image: str, paths to database images

    Returns:
        np.ndarray: Embeddings array (num_images, embedding_dim)
    """
    pass
```

All files should be uploaded to a single Hugging Face repository. Ensure that the function names, DataFrame column names, and file structure are consistent with the format outlined above.

Understanding generate_embeds.py

The `generate_embeds.py` file serves as a crucial standardized interface between your trained model and our evaluation system. The following diagram illustrates how `generate_embeds.py` fits into the evaluation process:



Evaluation Process

Evaluation Criteria

The primary metric for this challenge is Top-1 retrieval accuracy. For each query pair (image + text), your model should identify the correct matching image from the database. The accuracy is calculated as the percentage of queries where the highest-scoring database image (based on cosine similarity) matches the ground truth.

Evaluation Illustration

The evaluation process takes the embeddings generated by your model through the encode_queries and encode_database functions and computes the cosine similarity between them to identify the best matches. A visualization of this process is shown below:

```python
# 1. Load test queries and database
query_df = load_test_queries()
database_df = load_database()

# 2. Generate embeddings
query_embeddings = encode_queries(query_df)
database_embeddings = encode_database(database_df)

# 3. Calculate cosine similarity
similarities = cosine_similarity(
    query_embeddings,
    database_embeddings
)

# 4. Get top-1 predictions
predictions = np.argmax(similarities, axis=1)

# 5. Calculate accuracy
accuracy = calculate_accuracy(predictions, ground_truth)
```

After the challenge submission period ends, each team's best submission will undergo a final evaluation on a new hold-out dataset that has not been used in any previous evaluations. Final rankings and scores will be determined based on a combination of the test set used during the competition and the hold-out test set. Additionally, following the challenge, we will conduct a thorough code review of the top-performing teams' submissions. Teams will be disqualified if their code violates any of the specified constraints in this challenge, directly implements methods from published papers without significant original contribution, or contains pre-trained models or approaches that are explicitly forbidden. This comprehensive evaluation process ensures both the generalization capability of solutions and the integrity of the competition.

Key Notices

1. No internet access will be available during the test. If your code requires downloading any resources at runtime, your submission will be marked as failed. Please ensure that all necessary files and resources are included alongside your submission to avoid the need for any downloads. Make sure to test their code in an environment with no internet access and only the necessary packages installed to confirm that it doesn't require any downloads.

2. The entire evaluation of your submission will be conducted on a single Nvidia RTX 4090 GPU with 24 GB of VRAM. Please adjust your batch size in the `generate_embed.py` file to stay within this memory limit.

3. Your runtime should not exceed 10 minutes; otherwise, your submission will be marked as failed.

Important Links

- **Training Dataset**: [Dataset Link]
- **Evaluation Samples**: [Evaluation Samples Link]
- **Sample Submission**: [Sample Submission Link]

Steps to Set Up Your Environment

To ensure our judging environment matches yours, please provide your requirements file. Our environment runs on Python 3.10.11, so it is recommended to use the same version to avoid conflicts. You can download it from: Python 3.10.11 download. **Note: Please avoid using `conda` for creating environments, as it may lead to inconsistencies in the judging environment.**

1. **Install Virtualenv**:
   Install `virtualenv` by running:

   ```
   python -m pip install --user virtualenv
   ```

   If you receive a warning indicating `virtualenv` is not accessible, use the full path to the virtualenv script or add it to your PATH environment variable.

2. **Create Virtual Environment**:
   Create a new environment using Python 3.10.11:

   ```
   virtualenv [env-name] -p python3.10.11
   ```

   Replace `[env-name]` with your desired environment name.

3. **Activate the Environment**:

   - **Linux**:

     ```
     source /path/to/env/bin/activate
     ```

   - **Windows**:

     ```
     /path/to/env/Scripts/activate
     ```

   Once activated, the environment name will appear before your prompt.

4. **Install Requirements**:
   After activating the environment, install your required packages.

5. **Verify Python Version**:
   Ensure the Python version matches the judging system by entering the Python interpreter.

6. **Export Requirements**:
   Once all packages are installed, export them to a `requirements.txt` file:

   - **Linux**:

     ```
     echo "#$(python --version)" > requirements.txt && pip freeze >> requirements.txt
     ```

   - **Windows**:

     ```
     (<nul set /p=#&python --version & pip freeze) > requirements.txt
     ```