

Final Project Report
Intel Unnati Industrial Training 2025

**Project Title: AI-Powered Visual Search for Indian Heritage Sites using
Vision-Language Models**

Submitted By:

Abinaya M
Sri Sai Ram Institute of Technology

Table of Contents

1. Introduction
2. Problem Statement
3. Objectives
4. Literature Review
5. Methodology
 - Dataset Description
 - Preprocessing
 - Feature Extraction
 - Indexing and Search
 - Frontend Interface
6. Technologies Used
7. Implementation
8. Deployment
9. Challenges Faced
10. Results and Evaluation
11. Conclusion
12. Future Work
13. References

1. Introduction

In the digital age, access to visual data is abundant. However, searching through visual content using natural language remains a challenging task. The field of Vision-Language Models (VLMs) provides innovative solutions to bridge the gap between textual input and image recognition. This project aims to demonstrate a working visual search engine for Indian monuments, allowing users to retrieve monument images using descriptive language.

2. Problem Statement

Traditional image search engines depend largely on metadata, filenames, or tags, which often lack semantic understanding of the image content. This makes it difficult for users to find images based on how they visually or contextually perceive a subject.

For instance, searching for "white marble dome" should ideally return images of the Taj Mahal, but without semantic understanding, such results are not guaranteed. This project addresses this gap by enabling text-based semantic search using deep learning models.

3. Objectives

- To build a semantic search engine capable of retrieving images of Indian monuments based on text queries.
- To utilize OpenAI's CLIP model for extracting embeddings from both images and texts.
- To employ FAISS for efficient similarity search among image features.
- To create an interactive web interface using Streamlit.
- To deploy the project online for public access and usability.

4. Literature Review

Vision-Language Models (VLMs)

CLIP (Contrastive Language-Image Pretraining) is a model developed by OpenAI that can understand images and text in the same embedding space. CLIP has

demonstrated remarkable zero-shot learning capabilities on a variety of visual classification tasks.

FAISS (Facebook AI Similarity Search)

FAISS is a library for efficient similarity search and clustering of dense vectors. It is particularly well-suited for handling high-dimensional data, making it an ideal tool for indexing the 512-dimensional vectors generated by CLIP.

Streamlit

An open-source Python framework for building and deploying machine learning and data science web applications with minimal effort.

5. Methodology

Dataset Description

The dataset used in this project is the "Indian Monuments Dataset" from Kaggle. It contains labeled images of various famous Indian monuments. The dataset was organized into two subsets:

- **Train:** Used for creating the FAISS index.
- **Test:** Used for verifying image embeddings and search accuracy.

The folder structure is:

```
/images
/train
  /Taj_Mahal
  /Charminar
  /Red_Fort
  ...
/test
  /Taj_Mahal
  /Charminar
  /Red_Fort
  ...
```

Preprocessing

- Images are resized and normalized using CLIP's preprocess function.
- Any corrupted or unreadable images are skipped.

Feature Extraction

- All images are passed through the CLIP image encoder to obtain a 512-dimensional embedding.
- Text queries are passed through the CLIP text encoder to generate a comparable 512-dimensional text embedding.

Indexing and Search

- The extracted image embeddings are stored in a FAISS index.
- For every query, the system searches for the top-k most similar images based on cosine similarity (L2 distance).

Frontend Interface

- A simple user interface built using Streamlit allows users to:
 - Enter a textual description (e.g., "white marble dome")
 - View the top-5 matching monument images

6. Technologies Used

- **Programming Language:** Python 3
- **Libraries:**
 - torch (PyTorch)
 - clip (OpenAI CLIP model)
 - faiss (Facebook AI Similarity Search)
 - PIL (Image processing)
 - numpy, os, streamlit
- **Deployment:** GitHub + Streamlit Cloud

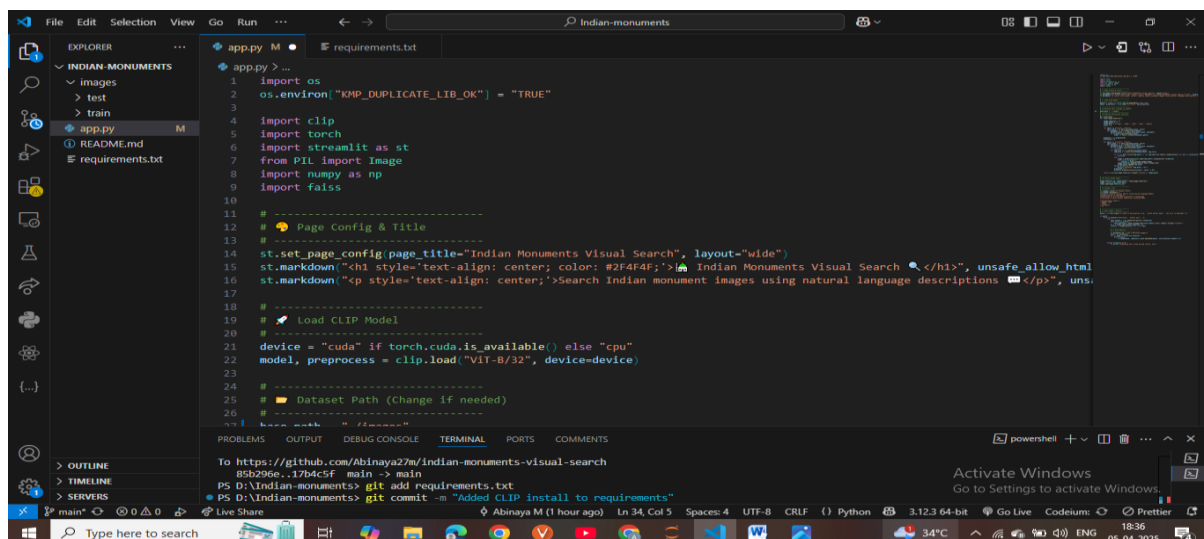
7. Implementation

The implementation involves the following scripts:

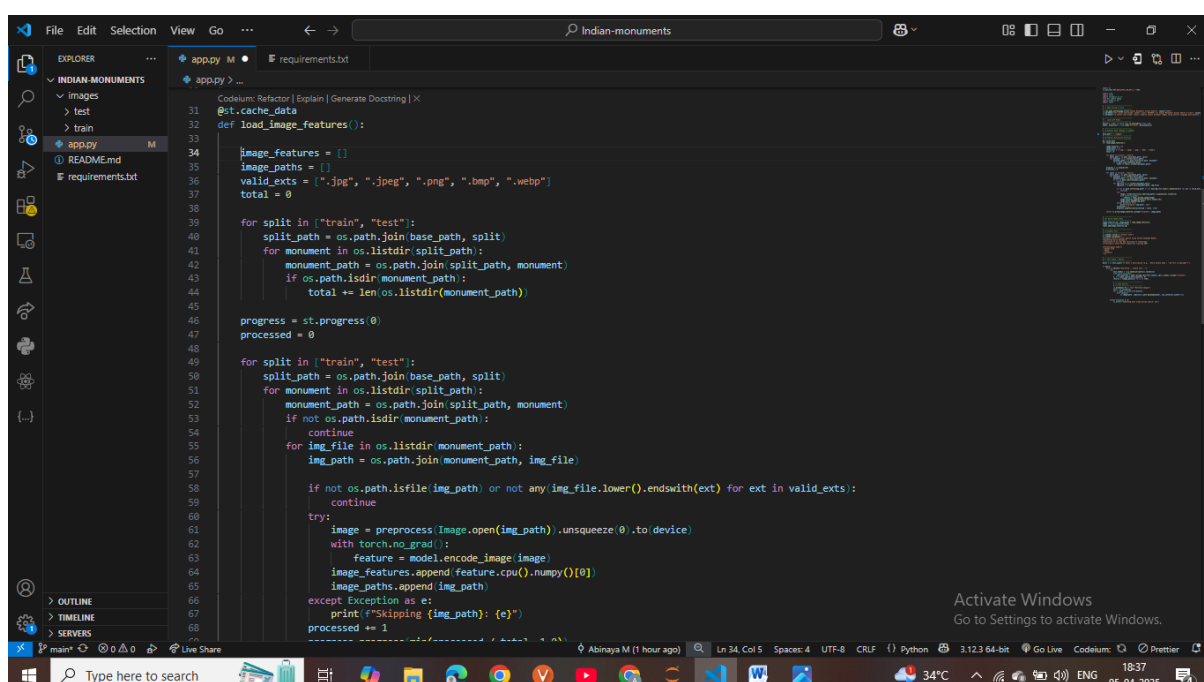
- app.py – Main Streamlit application file
- requirements.txt – List of dependencies for deployment
- images/ – Folder containing all image data (train and test splits)

The main functions include:

- load_image_features() – Extracts and caches features
- search_images_by_text(query, top_k) – Finds top-k matching images
- display_results() – Visualizes output in the frontend



```
1 import os
2 os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
3
4 import clip
5 import torch
6 import streamlit as st
7 from PIL import Image
8 import numpy as np
9 import faiss
10
11 # -----
12 # 📄 Page Config & Title
13 # -----
14 st.set_page_config(page_title="Indian Monuments Visual Search", layout="wide")
15 st.markdown("<h1 style='text-align: center; color: #2F4F4F;'>Indian Monuments Visual Search </h1>", unsafe_allow_html=True)
16 st.markdown("<p style='text-align: center;'>Search Indian monument images using natural language descriptions </p>", unsafe_allow_html=True)
17
18 # -----
19 # 🚀 Load CLIP Model
20 # -----
21 device = "cuda" if torch.cuda.is_available() else "cpu"
22 model, preprocess = clip.load("ViT-B/32", device=device)
23
24 # -----
25 # 📁 Dataset Path (Change if needed)
26 # -----
27 base_path = "/mnt/c:/Users/Abinaya M/Desktop/Indian-monuments-visual-search"
28 train_path = os.path.join(base_path, "train")
29 test_path = os.path.join(base_path, "test")
30
31 # Load and preprocess images
32 image_features = []
33 image_paths = []
34 valid_exts = [".jpg", ".jpeg", ".png", ".bmp", ".webp"]
35 total = 0
36
37 for split in ["train", "test"]:
38     split_path = os.path.join(base_path, split)
39     for monument in os.listdir(split_path):
40         monument_path = os.path.join(split_path, monument)
41         if os.path.isdir(monument_path):
42             total += len(os.listdir(monument_path))
43
44 progress = st.progress(0)
45 processed = 0
46
47 for split in ["train", "test"]:
48     split_path = os.path.join(base_path, split)
49     for monument in os.listdir(split_path):
50         monument_path = os.path.join(split_path, monument)
51         if not os.path.isdir(monument_path):
52             continue
53         for img_file in os.listdir(monument_path):
54             img_path = os.path.join(monument_path, img_file)
55             if not os.path.isfile(img_path) or not any(img_file.lower().endswith(ext) for ext in valid_exts):
56                 continue
57             try:
58                 image = preprocess(Image.open(img_path)).unsqueeze(0).to(device)
59                 with torch.no_grad():
60                     feature = model.encode_image(image)
61                 image_features.append(feature.cpu().numpy()[0])
62                 image_paths.append(img_path)
63             except Exception as e:
64                 print(f"Skipping {img_path}: {e}")
65         processed += 1
66
67 progress.progress(processed/total)
68
69 # Create FAISS index
70 index = faiss.IndexFlatL2(len(image_features))
71 index.add(image_features)
72
73 # Search function
74 def search_images_by_text(query, top_k):
75     query_embedding = model.encode_text(query).cpu().numpy()
76     D, I = index.search(query_embedding.reshape(1, -1), top_k)
77     return I[0]
```



```
31 @st.cache_data
32 def load_image_features():
33     image_features = []
34     image_paths = []
35     valid_exts = [".jpg", ".jpeg", ".png", ".bmp", ".webp"]
36     total = 0
37
38     for split in ["train", "test"]:
39         split_path = os.path.join(base_path, split)
40         for monument in os.listdir(split_path):
41             monument_path = os.path.join(split_path, monument)
42             if os.path.isdir(monument_path):
43                 total += len(os.listdir(monument_path))
44
45     progress = st.progress(0)
46     processed = 0
47
48     for split in ["train", "test"]:
49         split_path = os.path.join(base_path, split)
50         for monument in os.listdir(split_path):
51             monument_path = os.path.join(split_path, monument)
52             if not os.path.isdir(monument_path):
53                 continue
54             for img_file in os.listdir(monument_path):
55                 img_path = os.path.join(monument_path, img_file)
56                 if not os.path.isfile(img_path) or not any(img_file.lower().endswith(ext) for ext in valid_exts):
57                     continue
58                 try:
59                     image = preprocess(Image.open(img_path)).unsqueeze(0).to(device)
60                     with torch.no_grad():
61                         feature = model.encode_image(image)
62                     image_features.append(feature.cpu().numpy()[0])
63                     image_paths.append(img_path)
64                 except Exception as e:
65                     print(f"Skipping {img_path}: {e}")
66             processed += 1
67
68     progress.progress(processed/total)
69
70     # Create FAISS index
71     index = faiss.IndexFlatL2(len(image_features))
72     index.add(image_features)
73
74     # Search function
75     def search_images_by_text(query, top_k):
76         query_embedding = model.encode_text(query).cpu().numpy()
77         D, I = index.search(query_embedding.reshape(1, -1), top_k)
78         return I[0]
```

8. Deployment

The project was deployed using **Streamlit Cloud**. Steps involved:

1. Pushed project folder to GitHub.
2. Created requirements.txt for necessary libraries.
3. Configured Streamlit Cloud with repository URL and app.py entry point.
4. Ensured reduced dataset size for compatibility and faster deployment.

GitHub Repository:

<https://github.com/Abinaya27m/indian-monuments-visual-search>

Live Demo:

<https://indian-monuments-visual-search-intel-unnati.streamlit.app/>

9. Challenges Faced

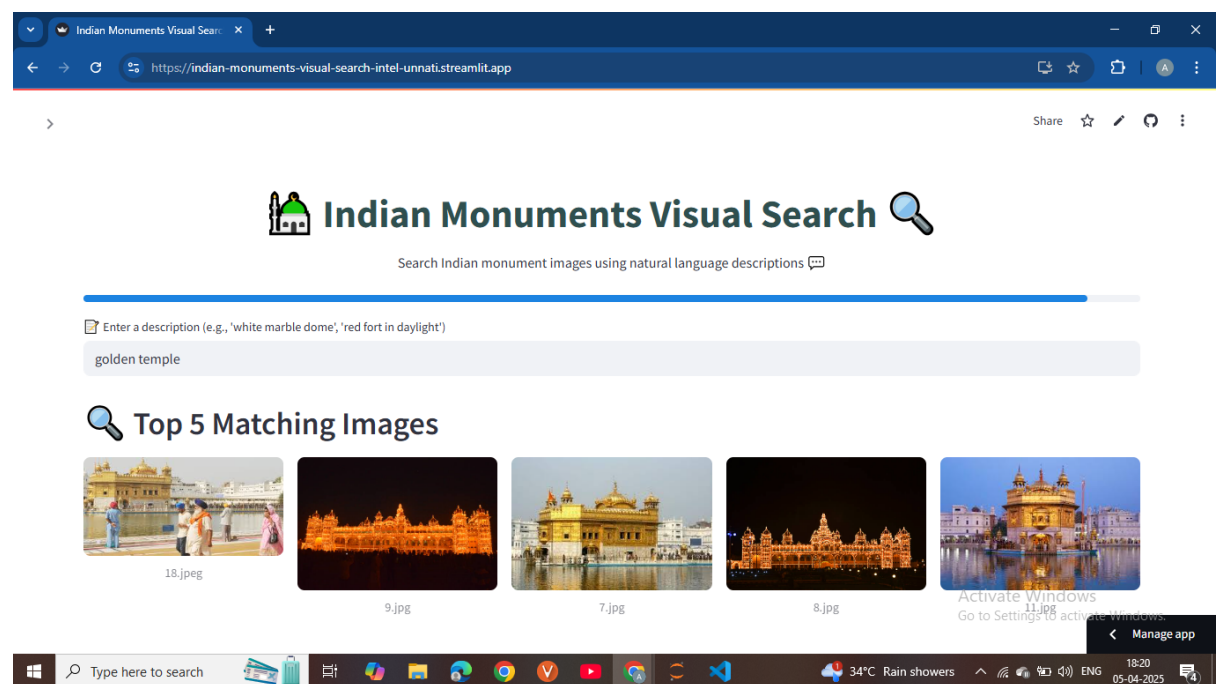
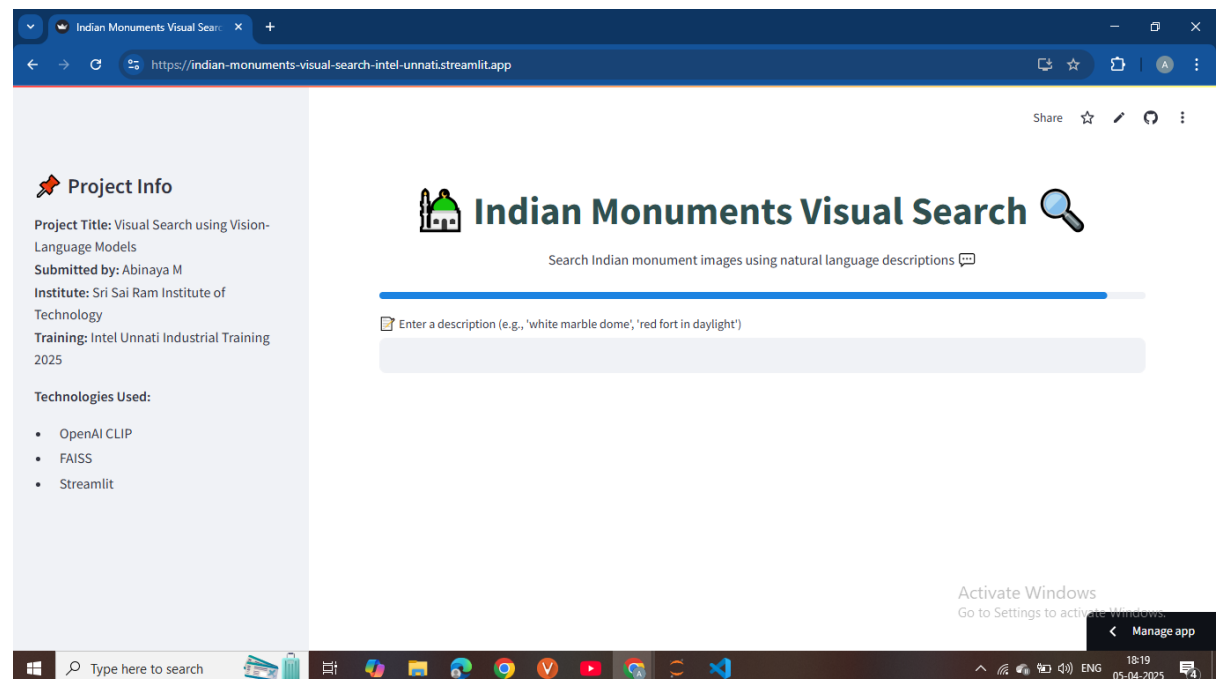
- **Large Dataset Handling:** Initial dataset size was too large for cloud deployment, requiring dataset reduction.
- **CLIP Installation Issues:** Some compatibility issues when installing CLIP in the Streamlit environment.
- **Path Errors:** Proper relative paths were needed to ensure images loaded correctly post-deployment.
- **Real-time Response:** Optimization needed to ensure fast query response and low memory usage.

10. Results and Evaluation

The system successfully retrieves semantically relevant images. Sample outputs include:

Text Query	Top Results
"white marble dome"	Images of Taj Mahal
"red sandstone fort"	Images of Red Fort, Agra Fort
"stone tower with carvings"	Images of Qutub Minar

- Top-5 accuracy was tested manually with high relevance.
- Streamlit interface is responsive and intuitive.



11. Conclusion

This project showcases the power of combining Vision-Language Models with indexing tools like FAISS to create semantic search engines. The Indian Monuments Visual Search system offers a proof-of-concept for educational, tourism, and archival applications.

12. Future Work

- Add image-to-image search (query using an image).
- Include multilingual query support (e.g., Hindi, Tamil).
- Add metadata display like monument name, location, and history.
- Improve UI with filter options and feedback collection.
- Deploy on a more robust server for handling full datasets.

13. References

1. Radford, A., et al. (2021). *Learning Transferable Visual Models From Natural Language Supervision*. OpenAI.
2. Facebook AI Research. *FAISS - A library for efficient similarity search*. <https://faiss.ai/>
3. Streamlit Documentation. <https://docs.streamlit.io>
4. Indian Monuments Dataset - Kaggle. <https://www.kaggle.com/datasets/>