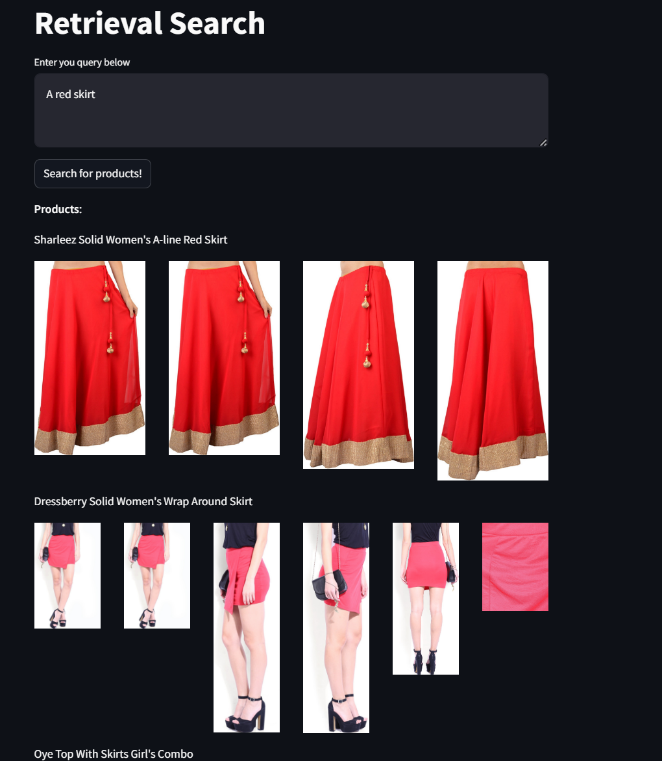
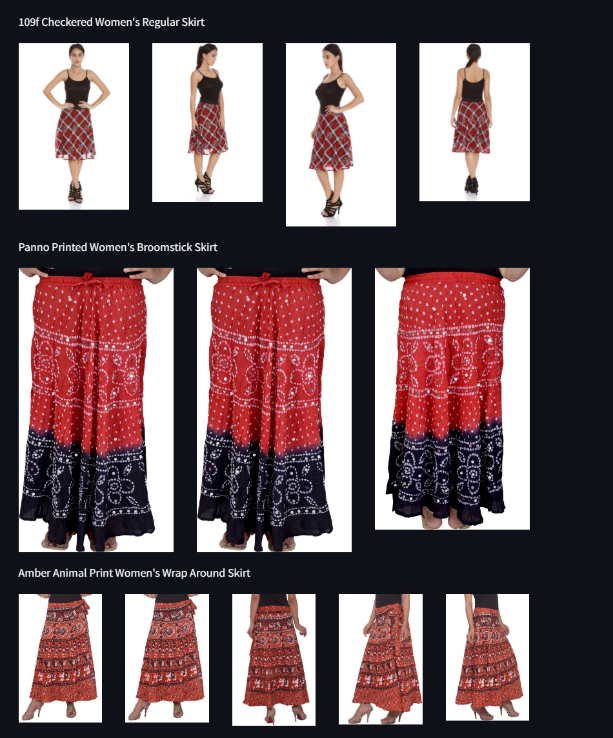
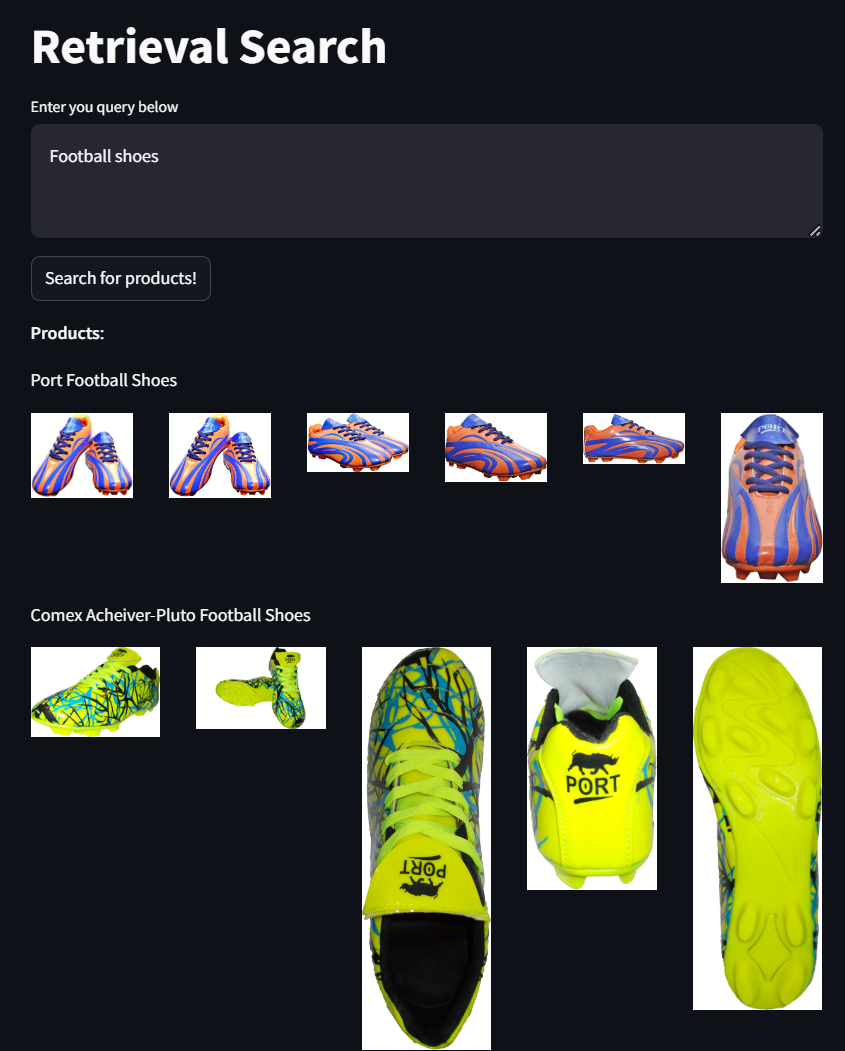
**Enhancing Search Experience for An E-commerce Store**

This is the doc detailing my approach for the task. Before diving in, let’s take a sneak peek at the final output.

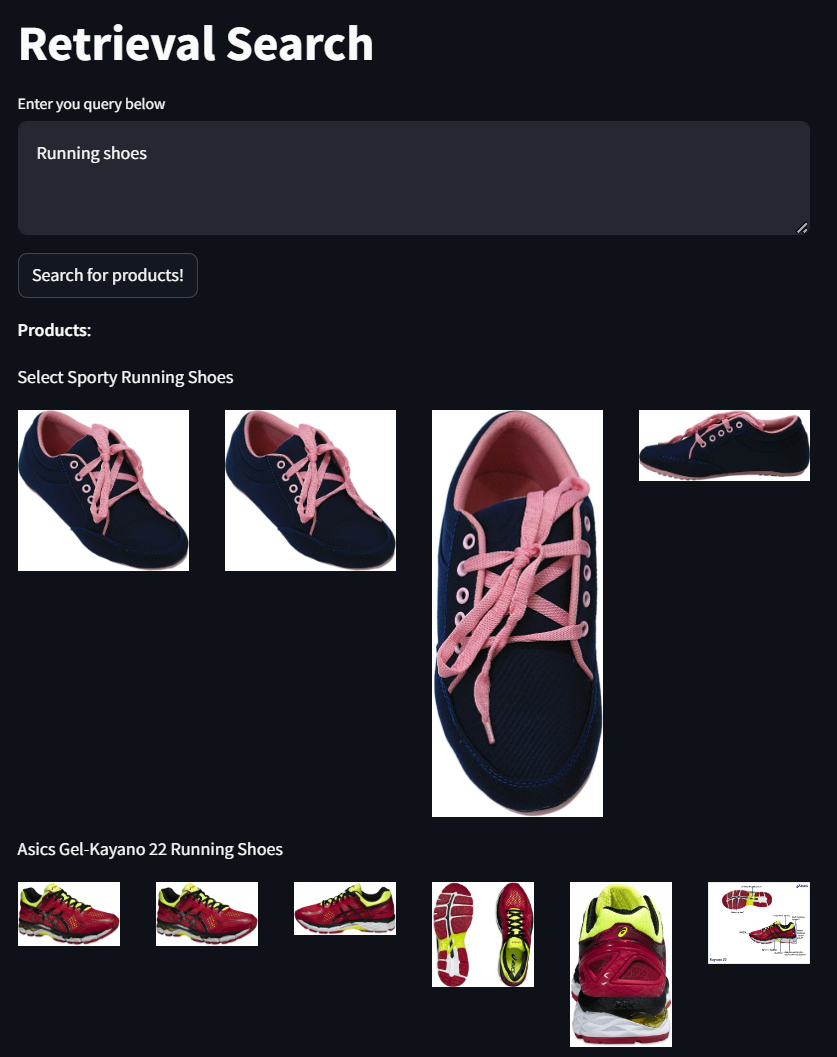
1. Query: A red skirt.

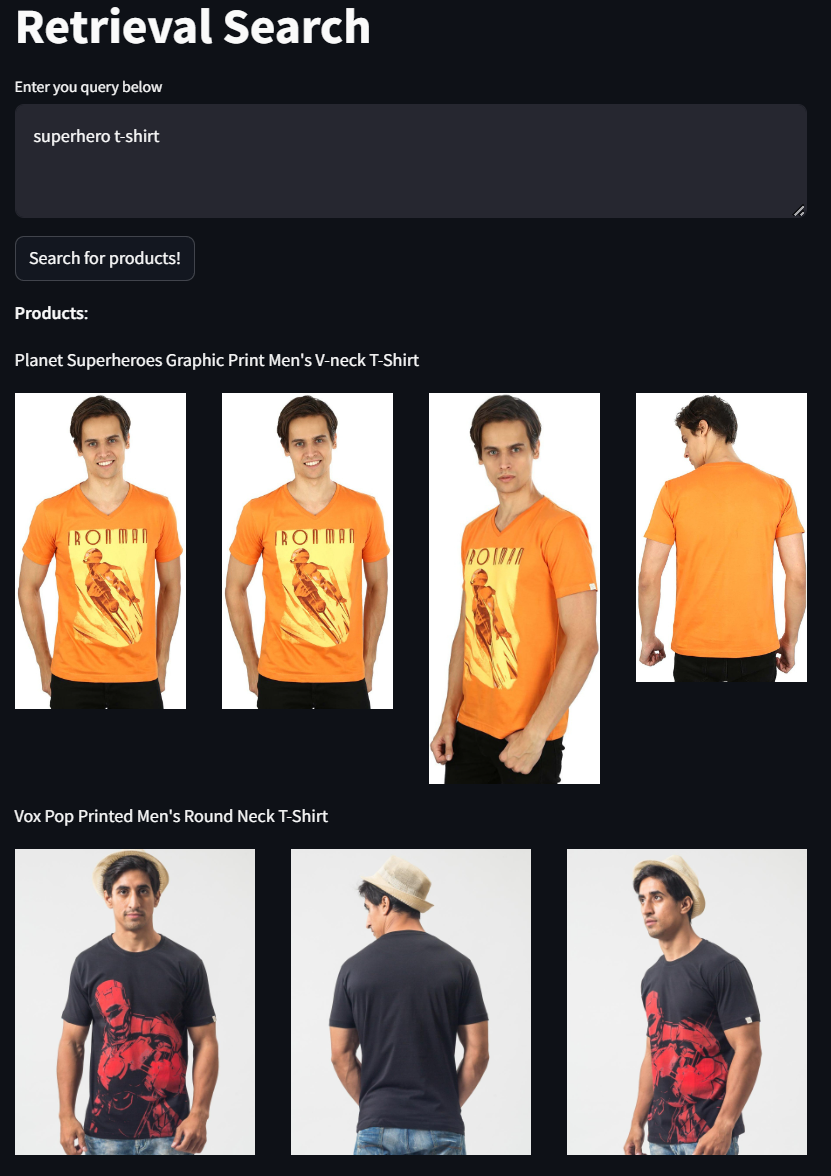
1. Football shoes

1. Running shoes

1. Superhero t-shirts

As you can see, the results are relevant to the search query. The model is able to catch even if the user query is specific or precise and demonstrated in the examples.

You can try out the app here: [App link](https://ecommerce-retrieval-search.streamlit.app/)

**Note**: The website may be a bit slow because it’s hosted on streamlit’s community cloud for now. The model to generate embeddings must loaded and run on CPU. So it may take time to retrieve results.

1. Exploratory Data Analysis
   * The goal was to get a good understanding of data, the text columns and the features I can utilize to build a good retrieval system.
   * I started with loading the dataset, examining all the columns, handling missing values, and gathering metadata of columns.
   * I analysed the most frequent categories of products and observed that there is a huge imbalance. Plotted a bar graph to visualize.
   * Then, I started looking into prices and analysed the relationship between actual price and discounted price. I observed that, at low prices there is not much difference whereas, as you higher up the prices the difference increases.
   * The next step was to look into the product and overall rating. Most of the rows here had no ratings, so this column was not that useful.
   * After all this, I concluded that for my task, the useful columns are
     + product\_name
     + product\_category\_tree
     + description
     + image
     + brand
     + product\_specifications
2. Preprocessing
   * Since most of our columns are text, preprocessing it is an important step.
   * I observed that product\_category\_tree and product\_specifications are not in the right form to be used directly. I preprocessed them so that they appear as natural text.
   * Now, I created a new text column by **concatenating** product\_name, description, product\_category\_tree, category\_specifications and brand in that order. This new text is used to build retrieval system.
   * This is because we have a way to utilize all the text for each product, which is beneficial because each product will become more feature rich due it’s unique description or product\_tree or it’s specifications.
   * After that, it’s time for text preprocessing. Removed HTML Tags, lowercased everything, removed punctuation and stopwords. Experimented with stemming, but it didn’t yield good results.
3. Embeddings
   * Now that I had my cleaned text, it was time to create embeddings. I searched through the ‘Massive Text Embeddings Benchmark’ and chose one which is light but has good performance. I settled with ‘gte-base-1.5’ embedding model, having 768 dimensions.
   * I converted 20,000 rows of text into embeddings efficiently using batch processing on GPU using Pytorch.
4. Similarity Search
   * Now that I had my embeddings, I needed a way to efficiently store and query them. Then comes, FAISS (Facebook AI Similarity Search). It’s a **Vector Database** used to do just that. I created a faiss index and stored my embeddings.
   * There was a choice, to use L2 distance, L1 distance or cosine similarity to query. Since FAISS just normalizes the distances, I tried L2 distance (Euclidean) and the results were satisfactory. I was not able to experiment much with other metrics due to time constraints.
5. Inference and GUI
   * The inference process was fairly simple. Get a user query, preprocess it in the same way to processed your earlier text. Collect it’s embeddings (this time on CPU), and query it on FAISS index and return the top K id’s.
   * Using the id’s retrieve the image from their respective URL’s and display them.
   * The next step was to create a quick, minimalistic GUI, which I did using streamlit. Finally, tested it and deployed it on streamlit’s community cloud.

**Tech Stack:**

* python – Primary Programming language.
* pandas – Handle csv files and dataframes.
* matplotlib – Visualizations.
* tranformers – embedding model
* torch – tensors, computing embedding and GPU support.
* nltk – stopwords, stemming (text preprocessing)
* faiss-cpu – Vector store
* streamlit – GUI and deployment.

**Future Experiments:**

* Due to time constraints, I was not able to experiment more. Off the top of my head, these are things which can be further done.
  + We could use a Vision-Language Pretrained model like CLIP, to collect image and text embeddings together, which could result in more better results.
  + This also opens up Image to Image, Image to text queries as well.
  + Another way is to fine-tune an Image model like (Resnet, ViT) on our images. This could result in better image embeddings than off the shelf models.
  + If there are multimodal user queries, we could use re-ranking to rank queries returned by image model and text model which improves results.
  + Collect a small subset of data by hand (manually), for different categories and evaluate the pipeline using metrics like Recall @ K etc.
  + Use an LLM to handle malformed and multilingual queries.