Multimodal Search in eCommerce

Tanuj Gupta[*] Meagan Kenney[†] Pavel Kovalev [‡] Chiara Mattamira [§]
Jeremy Shahan [¶] Hannah Solomon[*]

Abstract

There are many new powerful machine learning tools that have been recently developed that are useful for a searching engine. AI now has the capabilities of processing images and can quantify how similar any two images are. With this new application, people now have a new way to search the internet: image based searching. One of the biggest strengths of this new tool is that a customer does not need to know the name of what they are searching for. They can simply take a picture and use that in their search. However, one of the drawbacks of using purely image based searching is that a user cannot modify their search as easily as they could when doing a text based search.

This is the problem that we explore: how can we perform image based searching so that we can easily modify the search using text? We attempt to solve this problem in multiple ways: by using machine learning to match text-based queries to images, by combining this search with standard text-based searches, and by modifying the original image. To accomplish this multimodal search, we used tools such as CLIP, BM25, and SAM to build and evaluate several models that were then implemented in Streamlit. Specifically, we focused on modifying an image search of a fashion item with color. Combining image and text based searching in this manner opens up many new possibilities to change the way we navigate the internet.

1 Introduction

1.1 Motivation

Implementing multimodal search is intrinsically of interest to eCommerce companies, especially considering the remarkable growth of the eCommerce sector. Forecasts predict a staggering 443% increase in sales between 2014 and 2025, showcasing the industry's steady expansion [40]. Notably, about 50% of retailers are planning to increase their investments in website search to cater to customer needs effectively [1].

With regards specifically to eBay, it controls a large portion of eCommerce with a total of 1.6 billion live listings across 190 markets [39]. As one of the top eCommerce sites globally, eBay is well-positioned to capitalize on emerging trends and customer demands. Furthermore, Amazon and Google have already developed effective tools, namely Amazon Lens and Google Lens, to address the aforementioned customer problem. Being part of the same industry adds to the list of reasons why eBay would greatly benefit from developing a similar tool.

1.2 Goal

Our initial goal of the project was to create a search method incorporating both image and text queries. In particular, we wanted to implement a multimodal search that could take an image query of a clothing

[*]Texas A&M University, College Station
[†]University of Minnesota, Minneapolis
[‡]Indiana University, Bloomington
[§]University of Tennessee, Knoxville
[¶]Louisiana State University, Baton Rouge

item and a text query specifying how the user would like to modify the image, such as color, and return results similar to the image query, but with the modifications made. Tools such as CLIP and the Segment Anything model (discussed in Section 2) that can process images are helpful in accomplishing this goal. We strived to gain an understanding of these tools in order to incorporate them into our search method to improve its accuracy. Furthermore, we wanted to seek out additional tools and methods to utilize in our search method and compare the accuracy of those models with the initial ones.

1.3 Our dataset

The dataset used in the project [22] was made available by Myntra, an Indian fashion eCommerce company. It consists of 44,424 fashion products for men, women, and children. Due to missing a title description or image, 12 products were excluded from the dataset. All other products have a caption describing the item and an associated picture. Moreover, each product has an id and is classified based on gender, article type, color, season, year, and usage.

About half of the products fall under the broad category of apparel, with t-shirts, shirts, and shoes being the most popular items. The dataset contains a roughly equal ratio of products for men and women, and both categories are represented approximately three times more than the children's categories (boys and girls). Furthermore, since the focus of this project is color, it is worth mentioning that the most common colors in the dataset are black, white, blue, brown, gray, red, and green. On the other hand, yellow and orange are the least popular primary and secondary colors respectively.

Finally, it's worth noting that the size of our dataset is significantly smaller than the size of eBay's. This poses some limitations when searching for products that don't appear much in the dataset. For example, in our work related to changing patterns, we were unable to test how well searching for a leopard print shirt does due to the lack of such products in the dataset.

2 Tools Used

2.1 Contrastive Language-Image Pre-Training

2.1.1 How it works

Contrastive Language-Image Pre-Training (or CLIP) [27] is a deep learning algorithm from OpenAI that uses natural language supervision to transform images and text into vectors that lie within the same space. Compared to previous AI models that focus solely on language, this better encompasses the multimodal world in which we live, where text is not the only means of communicating ideas [29].

CLIP comes equipped with two encoders – a 12-layer transformer with 63 million parameters for encoding strings of text and a vision-like transformer ("ViT") together with a 50-layer convolutional neural network for encoding images [33, 6]. Specifically, we used the version of CLIP from Hugging Face *Transformers* [8]. It includes a CLIPTokenizer that can be used to encode text alone and a CLIPProcessor that can encode both text and image. The CLIPProcessor also pre-processes the images by splitting them up "into a sequence of fix-sized non-overlapping patches, which are then linearly embedded," as well as rescaling and normalizing the images [6]. While the CLIPProcessor can handle an empty string of text, for cases where just the image needs to be encoded, it does not work with an empty image, for cases where just the text need to be coded. In these instances, CLIPTokenizer is used on its own, which encodes text the same way that CLIPProcessor does [6].

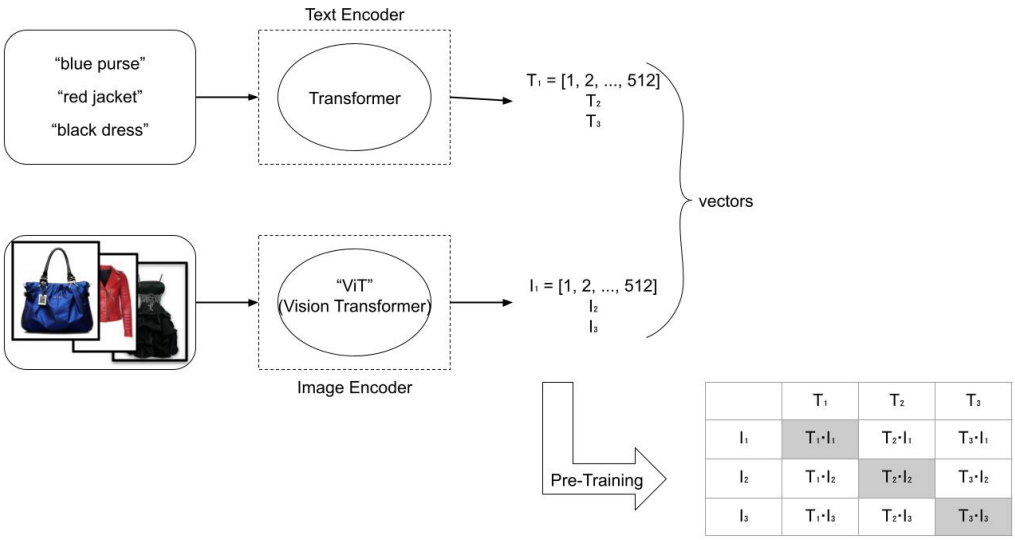

Figure 1: Encoding and Pre-Training Process. Note that the gray cells should be maximized and the non-gray cells minimized. Image Sources: [13, 19, 10]

Both of these encoders construct a 512-dimensional vector representing what is described by the words and depicted by the picture. They are not encoding pixels, characters, or any other sort of metadata, but rather focused on the content of the images and text. The model was trained on a large, diverse dataset of about 400 million images with corresponding text descriptions, sourced from the internet, to learn patterns - "for language models, that may be the general rules and patterns in the English language, while "for vision models that may be the characteristics of different scenes or objects." [29, 33]

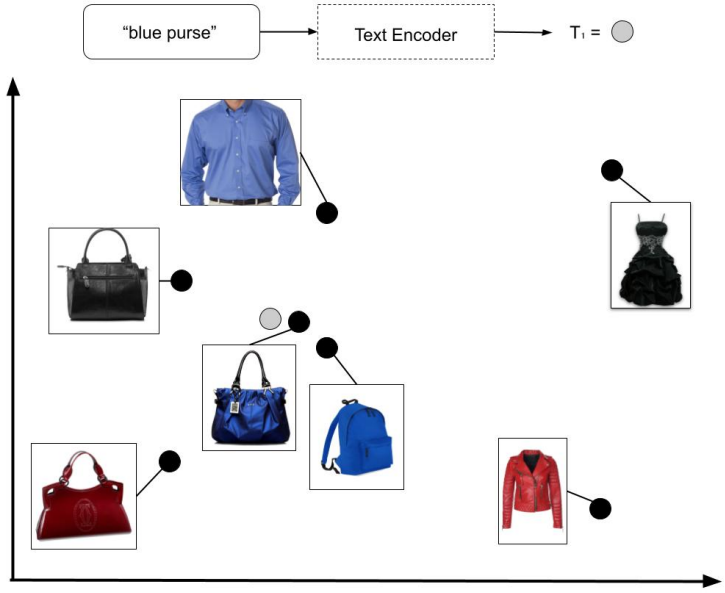

Figure 2: Vectors in Space. After transforming the text "blue purse" into T_1, we can compare it to a variety of images. Although we are comparing text to image, since the vectors are in the same space, we can see that the images most similar to a blue purse have vectors near T_1. Image Sources: [13, 11, 20, 12, 19, 10, 14]

By transforming from two separate modes to the one mode of vectors, CLIP provides us with a means to compare text to image. Ideally, similar images and text would have vectors near each other and dissimilar ones far apart. After encoding the text and image information into vectors, the image vectors could easily be compared to other image vectors, and similarly with the text vectors, but the model did not yet have a way to understand the similarity of a text vector with an image vector [29]. For this to work, the model was pre-trained using batches of 32,768 pairs of text and image, which were fed in parallel into the encoders [33]. The vectors produced were then compared (after normalization) using a dot product. Since the pre-training wanted to not only maximize the vectors for an image and its text description but minimize the similarity between non-corresponding pairs, a loss function was applied to the dot product. This is called contrastive pre-training for the way it compares and contrasts the similarity. Now, CLIP is a model where "we can search across any modality with any modality" [29].

2.1.2 How we are using it

In our case, we applied CLIP to our dataset of 44,412 fashion items, each with an image and title. Rather than needing to call on CLIP each time we wanted to apply it to the dataset for comparison, we stored the vectors for each image and for each title in image and title tensor files, respectively. A user could then upload an image or enter a string of text, which could be encoded using CLIP and compared against the vectors from our dataset. With CLIP, we have lots of options.

- (image-to-image): A user uploads an image and we compare it to the images in the dataset.
- (image-to-text): A user uploads an image and we compare it to the titles in the dataset.
- (text-to-image): A user types in a string of text and we compare it to the images in the dataset.
- (text-to-text): A user types in a string of text and we compare it to the titles in the dataset.

The different results are illustrated in Figure 3.

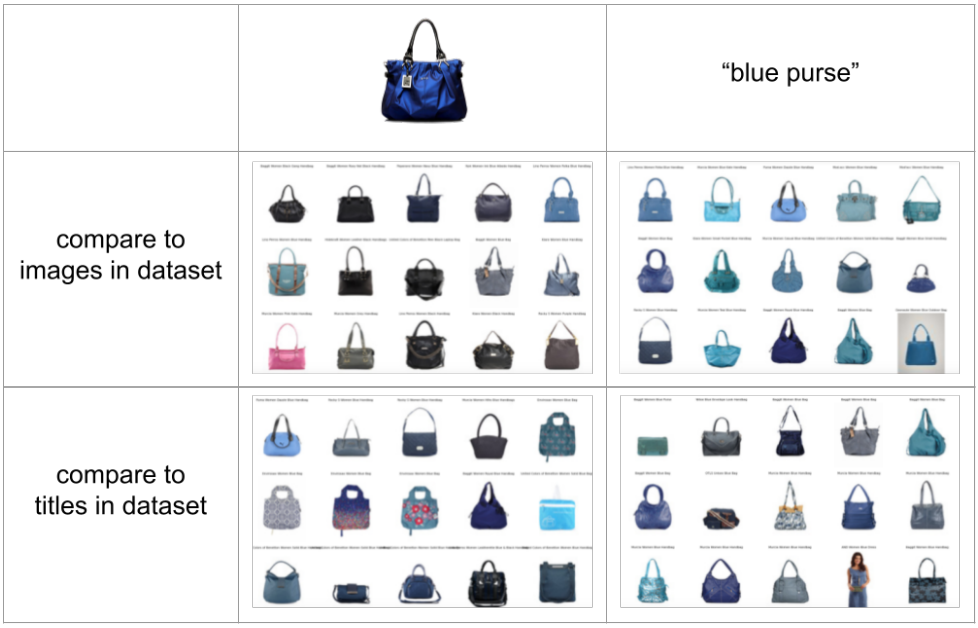

Figure 3: Comparision of different search results. Image Source: [13]

We found due to the inconsistency in titles, text search worked best when compared to the images rather than the text in our dataset. In Section 2.3, we detail methods used for comparison, whether

it be image-to-image or text-to-image. This gives us two separate modes of search and in our models (detailed in Section 3), we combined them to create multimodal search.

2.1.3 Strengths and weaknesses

One of the undoubtful advantages of CLIP is that it allows for semantic search across any modality with any modality: the input can be either an image, or a piece of text, or both, and the output can be either an image or a text. Thanks to the contrastive pretraining, this search is content-based, as opposed to being based on exact matches or attached metadata.

Another important advantage of CLIP is that it is zero-shot adaptable. This means that zero samples are required in order to train the model to perform well on a new dataset. In other words, CLIP performs quite well on new datasets without the need for fine-tuning.

The main drawback of CLIP is about searching with specific words. If you want the results to exactly match a particular word, CLIP might find it challenging. For example, if you search for "Adidas blue shoes for men" in an e-commerce database that contains images and titles of products, CLIP results may include shoes for men, but it may not show a lot of Adidas brand shoes. For a more extreme example, if the brand name is "Do U Speak Green" and the user searches for "Do U Speak Green Men Pants", then CLIP is likely to deduce from this query that relevant results should contain men green pants, as it is not obvious that "Do U Speak Green" is merely a brand name.

2.2 BM25

2.2.1 How it works

One of the important search algorithms we have used is called BM25. The BM stands for best matching and is a text-based search algorithm. Given a list of documents, BM25 searches through each document and tries to return the most relevant documents when given a text query to search by. In our case, the documents are the product descriptions coming from the titles of the images. The algorithm works by taking the input text query and splitting it up into a list of words. Then, for each individual word, the algorithm goes through each document and counts the number of times the word appears and assigns the document a score for that particular word. For each document, all these scores are added up resulting in one overall score for each document. The most relevant documents are those with higher scores.

While calculating a score for an individual document, there are several factors that BM25 takes into account: the frequency each word appears in the document, the number of documents in which each word appears, and the length of each document.

The first factor is easy to understand; if a word that you are searching for appears frequently in a document then BM25 will output higher scores than for documents in which that word does not appear as frequently. However, not all words are weighted equally. BM25 assigns weights by inverse document frequency meaning that if one of the search words appears in many documents then BM25 will assign a lower weight to that score. In other words, if a word appears frequently throughout many documents, then BM25 considers this to be a less important word. Words such as "a", "the", "and", etc. appear very frequently and are assigned lower weights and have less impact on the overall score. Additionally, the length of the document is considered as well. Documents containing many words have better chances at containing one of the words from the text query and are penalized.

Since each product has both an image and a description associated with it, we can search through all the descriptions and score products based on how well the description matches the text-based query.

We use these BM25 scores in conjunction with other search methods to create an overall score for each product.

2.2.2 Strengths and weaknesses

As mentioned above, BM25 checks to see if any of the words from the text query appear in the document. This algorithm checks for exact word matches and will not account for synonyms, misspellings, and alternative spellings. Another downside of BM25 is the way that it assigns weights. For example, the word man appears very frequently in the dataset and is assigned a small weight, but to a user it may be very important in their search if they only want products for men.

Despite these weaknesses, if managed well, BM25 is still very useful and can produce many relevant results. One of its core strengths is that it is very efficient and produces results quickly minimizing computation time. It is fast, efficient, and effective which has resulted in widespread use in searching engines.

2.3 Similarity search

2.3.1 Introduction

After obtaining text and image embeddings using CLIP, we need to perform a similarity search across these vectors. Traditional databases typically store data in tables containing metadata. For instance, an image database might have a table where each row represents an image, and the row includes information such as description, title, etc.

However, traditional query languages like SQL are not well-suited for performing similarity searches on vector data, as all features are encoded in vectors in an uninterpretable manner, making standard key-value queries impossible. To address this limitation, additional libraries are needed to perform efficient similarity searches. While there are several libraries available, we specifically utilized Faiss and Pinecone. In the following sections, we will discuss these libraries in more detail.

2.3.2 Background: sparse and dense vectors

In order to prepare the ground, we need to take a brief detour to introduce the terminological distinction between sparse and dense vectors. Sparse vectors typically have tens or hundreds of thousands of entries, with most of these entries being zero. They are good at syntax-based comparisons of two sequences, making them suitable for identifying similarities based on word arrangements. For example, when comparing two sentences with different meanings but similar syntax, sparse vectors can produce a close or even perfect match. However, sparse vectors are not particularly suitable for semantic search [28].

One of the algorithms used for generating sparse vectors is BM25, which we discussed in Section 2.2. This algorithm adopts the traditional bag of words (BOW) approach, where each document (or title, in our context) is transformed into a set (or “bag”) of words. The resulting data is then used to populate the sparse “frequency vector” [32].

Unlike sparse vectors, dense vectors have significantly fewer entries and the majority of these entries are nonzero. While dense vectors are not interpretable, they effectively capture the meaning of the data. CLIP text and hybrid encoders (see Section 2.1) serve as examples of methods for generating dense vectors.

To sum up, sparse vectors are more suitable for syntactic-based search, whereas dense vectors are more suitable for semantic-based search. The natural question that arises is how to combine the two in order to obtain the best results possible. The traditional approach to combine dense and sparse vectors involves a two-stage retrieval and reranking system. Initially, we perform similarity search using either a dense or a sparse vector alone, and then rerank the results based on the other type of vector. However, this method has several disadvantages: slower retrieval process, increased engineering challenges, and its effectiveness heavily relies on the initial retrieval stage's accuracy [32].

Luckily, there are powerful vector databases that allow both types of vectors to populate a single hybrid index and perform a single-stage retrieval, which can potentially improve the results. An example of two-stage and one-stage retrieval systems is given in Figure 4.

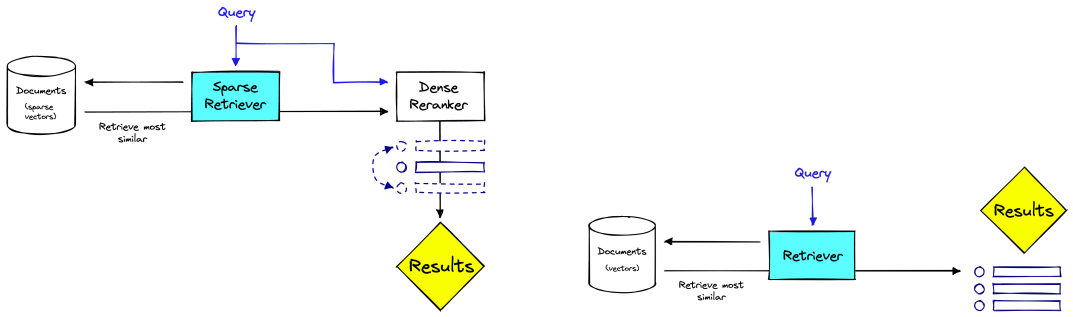

Figure 4: Two-stage retrieval vs. one-stage retrieval [32]

Before discussing one database that allows for hybrid indices that we used, namely Pinecone, we describe a more basic index called Faiss that we used in many models and point out its limitations that can be overcome with the use of Pinecone.

2.3.3 Faiss

Facebook AI Similarity Search, or Faiss for short, is a library that facilitates efficient similarity search. In essence, it enables users to index a set of vectors, and subsequently, given a query vector, it efficiently searches for the most similar vectors within the index. There are many different types of indices available in the Faiss library, which have their own advantages and disadvantages [4, 30]. We have only used the flat $L2$ index, and we describe it below.

Flat indices simply convert the vectors into fixed-size codes and store them in an array. These indices are referred to as "flat" because the input vectors remain unmodified. During search, all the indexed vectors are decoded one by one and compared to the query vectors [4]. After these comparisons, the k nearest neighbors are returned.

Because these vectors remain unchanged, the flat index provides highly accurate results. However, this in general comes with the trade-off of longer search times [30]. Luckily, since our dataset is not very large, the search time was not a major concern, and even with the flat index, the search was conducted relatively quickly.

One of the drawbacks of Faiss is that it is problematic to use it to create indices populated by sparse vectors. This is due to their extremely high dimensionality. Our attempt to create a sparse index using Faiss encountered RAM limitations, as Faiss restricts memory usage to 30 GB. A potential workaround is to store only the nonzero entries of sparse vectors along with their corresponding indices, like how Pinecone handles sparse vectors as dictionaries. Unfortunately, Faiss does not support populating in-

dices with dictionaries, which presents a limitation in utilizing this approach. Additionally, Faiss does not support hybrid indices.

2.3.4 Pinecone

Pinecone is a vector database, purpose-built to efficiently manage vector embeddings and offers several advantages over standalone vector indices like Faiss. It provides easier data management with features for storage, updates, and deletions, supports metadata storage for finer-grained queries, ensures scalability for handling large datasets, allows real-time updates without re-indexing, offers seamless integration with data processing ecosystems, and provides built-in data security and access control features [34].

What singles out Pinecone from other vector databases is that Pinecone makes it possible to perform hybrid search using a single hybrid index, letting us overcome the limitations of Faiss mentioned in Section 2.3.3. It allows storing both dense and sparse vectors in a single index, thereby reducing the two-stage retrieval and reranking system to a single-stage retrieval, as discussed in Section 2.3.2.

During the querying process, the input should be a vector containing both dense and sparse values obtained from the query. Pinecone ranks the vectors in the hybrid index by computing the dot product between the dense and sparse parts of the query with the dense and sparse parts of every vector in the index. The final score of a vector is determined by summing up the dot product of its dense values with the dense part of the query, along with the dot product of its sparse values with the sparse part of the query [31].

Moreover, with Pinecone's implementation of hybrid index, it is straightforward to assign weights to dense and sparse vectors. We followed the official Pinecone documentation [31] in using the following formula for assigning weights:

$$\text{hybrid score} = \text{alpha} \times \text{dense} + (1 - \text{alpha}) \times \text{sparse}$$

By setting alpha to 1, the search exclusively relies on dense vectors, transforming it into a purely semantic search. Conversely, when alpha is set to 0, the search solely utilizes sparse vectors, converting it into a syntactic search. Tuning the alpha value allows for achieving optimal results by balancing the influence of dense and sparse vectors in the search process.

Pinecone has a couple of disadvantages, some of which are related to the limitations of the free plan. For example, with the free account, one can only create one index, which may make experimentation challenging. Moreover, since indices are stored on Pinecone servers, it is impossible to share one's index without also sharing one's API key. While these limitations might not be a significant issue for smaller-scale projects like ours, they can become more noticeable when dealing with larger-scale applications. However, for such large-scale projects, the company may find it feasible to invest in the paid version of Pinecone.

2.4 Segment Anything

With the increasing popularity of visual search in eCommerce, which allows users to search for items by images taken with their mobile devices or uploaded pictures from their photo library, there is a high demand to improve visual search capabilities. A useful tool in doing this is the state-of-the-art image segmentation model called Segment Anything Model (SAM) released earlier this year by Meta's FAIR (Fundamental Artificial Intelligence Research) lab. This model has been trained on a large amount of data (images and masks) and does not require additional training. SAM is comparable to other

fine-tuned models [23] and is trained to be a zero-shot learning model, which means it can perform segmentation well on images it has not been trained on.

2.4.1 How it works

The architecture of SAM comprises three main components: an image encoder responsible for computing one-time image embeddings, a prompt decoder that embeds the provided prompts, and a lightweight mask decoder that combines the embeddings from both the prompt and image encoders to predict segmentation masks.

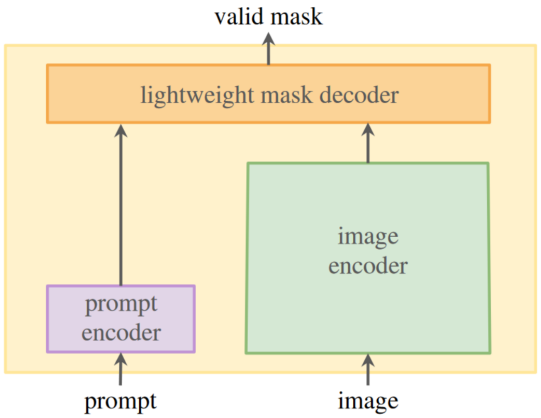

Figure 5: Architecture of SAM [23]

The image encoder is implemented in PyTorch and requires a GPU for efficient inference [37], while the prompt encoder and mask decoder can run directly with PyTroch or converted to ONNX and run efficiently on CPU or GPU across a variety of platforms that support ONNX runtime.

SAM is a transformer-based deep learning model trained on over 1 billion masks and 11 million images. Both the model and the dataset (SA-1B) have been released as open source on their website [37] for research purposes. SAM is designed and trained to be promptable, enabling it to perform image segmentation on any given image using prompt engineering.

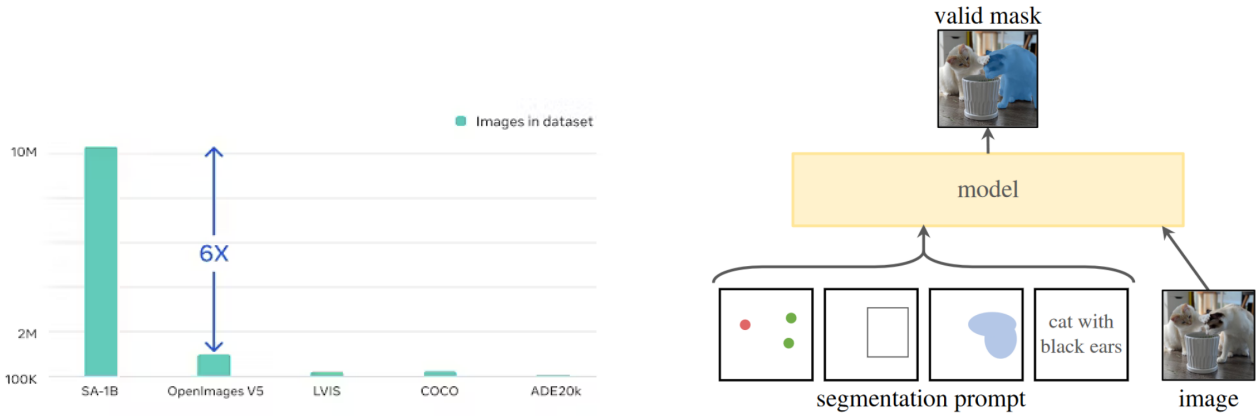

Figure 6: Left: comparison of training data across different models; Right: SAM's task [23]

The prompts serve as instructions to guide SAM in segmenting the image based on the user's requirements. These prompts can take the form of single or multiple points on the image, a bounding rectangle

around the object of interest, or even a descriptive text specifying the object the user wants to segment. While the paper explores and demonstrates text prompts, this capability has not been released as of today.

The result of SAM's segmentation is an image of the same dimensions as the original image, which retains the colors of the pixels contained in the item of interest, but assigns a white color to every other pixel.

Figure 7: Left: original image [15]; Right: segmented image

2.4.2 How we are using it

We utilize two prompts – a bounding box and a point on the image. The bounding box helps SAM focus on segmenting the part of the image that contains the item of interest. Subsequently, we select a point on the image that corresponds to the item of interest. These prompts enable SAM to perform segmentation according to our specific requirements.[1]

2.4.2.1 Removing unwanted items from the image In [2], the authors conduct a search log analysis of over 1.5 million image queries issued to the mobile application of eBay over a four-week period. They discovered that approximately 80% of the image queries consisted of images taken with the device's camera. One of the primary challenges with visual search using camera images is that the background may contain items that can negatively impact the search results. For instance, if a photo of a model wearing a blue shirt and red pants is used for visual search, the results might include both blue shirts and red pants. It may even include red shirts and blue pants as depicted in Figure 8. While this problem could be resolved by cropping the image in simple cases, it is not always feasible. Therefore, we employ the Segment Anything Model (SAM) to isolate the desired item and subsequently perform visual search using the segmented image, resulting in significantly improved outcomes.

[1] It's worth mentioning that in the official GitHub repository [38], the `predictor_example.ipynb` file contains code that uses both a bounding box and a point prompt, but they use the point prompt to exclude the item from segmentation. The `input_label` for the point in the `predictor_example.ipynb` is 0, but we changed it to 1 for our purposes, since we wanted to include the marked item.

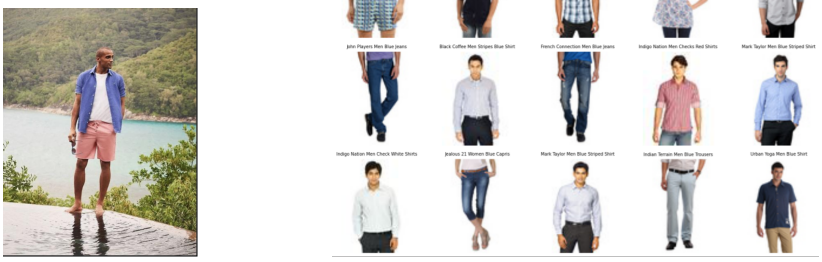

Figure 8: Left: original image; Right: image search result using original image

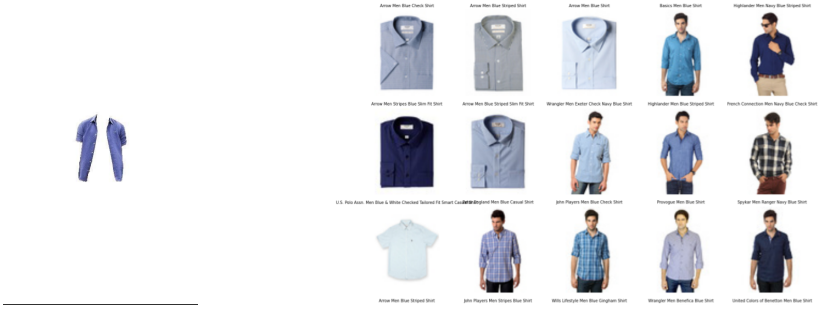

Figure 9: Left: segmented image; Right: image search result using segmented image

2.4.2.2 Changing the color and fabric pattern of a clothing item In a multimodal search, users can enter both a text and an image query to refine and enhance the search results. To improve the output of CLIP's image embedding we set out to alter the image itself with the user's text query in order to make a new image query. This new image query now gives an image embedding that is more similar to the search results we are hoping to see. Using SAM, we create a mask of the clothing item of interest in the image query. Isolating this item allows us to identify the pixels that require a color change. We calculate the average color of the item and find the differences in RGB values between each pixel and the average color. This helps retain important details like shadows and seams during the color change. For each pixel, we can then change the color to the RGB value of a new color, accounting for these details. However, one issue arises with the RGB value change, as it can produce colors outside the acceptable range of 0 to 255, resulting in unnatural and undesirable hues in the modified image. To address this problem, we rescale the new pixel values to fit within the appropriate range, leading to successful color changes.

Figure 10: Left: original image [17]; Right: altered image (changed to (100, 149, 237))

The altered photo can now be used as an image query in our search models. Beyond color changes, we

also implemented pattern alterations. The simplest approach was overlaying a new pattern image onto the mask of the clothing item. To support this feature, we created a database of some of the 20 most common fashion patterns, including stripes, check/singham, polka dots, and leopard print. Applying the same method as for color changes, we preserved the original details on the newly patterned clothing item. For example, this image demonstrates the color and pattern change process on an LL Bean shirt.

Figure 11: Left: original image [16]; Right: altered image (dark green check pattern)

2.4.3 Strengths and weaknesses

SAM does a great job of creating masks for the item of interest from an image file, significantly improving the quality of the search results. However, contrary to the claims on Segment Anything website, we found that performing segmentation using SAM on our computers takes a while, with an average processing time of roughly 1-2 minutes. This delay could be attributed to inefficient implementation of SAM on our end.

One major weakness of using SAM in our project is that if the item of interest is being blocked, for example, by the hand of the person in the image, then the segmented item will lose its natural shape and this can negatively impact the CLIP image search. Figure 12 is an example where the image search of the segmented image of a black dress results in items such as socks and ties.

Figure 12: Left: original image [18]; Middle: segmented image; Right: search results using segmented image

In addition to the limitations faced by SAM in general, the current method for pattern changing does not always work for altering the pattern of an already patterned clothing item. This is due to the fact

that we rely on the difference in colors in the original image to retain the shadows, seams, and other details. If the original image has a pattern, for example black and white stripes, the current method would interpret the black stripes as being the same as a shadow or seam and thus would bring it onto the new image creating a stack of patterns, which is not desirable.

2.5 Streamlit

Streamlit is a Python library that helps create and deploy custom web apps for machine learning and data science. It has a wide variety of widgets that are particularly suitable for our purpose, such as file uploader, text input, slider, and color picker. It is relatively straightforward to wrap the existing code around the Streamlit code, making the app creation process quite easy.

However, there's a limitation when deploying apps on the Streamlit Community Cloud. Implementing SAM in our app requires access to a large vision transformer model file, which necessitates the use of GIT-LFS to upload it to [42]. Streamlit has difficulty handling this file through GIT-LFS.[2] Because of this, we couldn't deploy the full app on Streamlit's platform and had to host it on our own computer. You can watch a demo at this link: `https://youtu.be/2GWWDFh-3QA`. In the video, we demonstrate how a user might use SAM to pick out an object from the image and then change its color and pattern. We then take the modified image and perform a multimodal search on our data set yielding some good results.

Although we haven't been able to deploy the full version of the app on the cloud, we have been successful in deploying a limited version not including SAM. This version of the app is available at `https://visualsearchbasicdemo.streamlit.app/`.

3 Models used

With the tools detailed in Section 2, we are ready to construct models that combine the different methods together and allow for multimodal search. By using multiple tools, we can address some of their weaknesses and optimize our models to play to their strengths.

Summary of Tools			
Tool	**image**	**text**	**Description**
CLIP	✓	✓	uses description and depiction to encode semantic meaning of images and text
BM25		✓	searches for best match syntactically
SAM	✓		alters segment of image using mask

Figure 13: Summary of tools used

3.1 Overview of models used

We used three basic models which are summarized in Figure 14 below. Each of these models in fact yields an infinite number of models since the weights can be adjusted.

[2] In particular, we tried uplaoding the smallest transformer file `vit-b` to run SAM on Streamlit but it doesn't work [41].

Model 1	CLIP Image + CLIP Text with Faiss
Model 2	CLIP Image + CLIP Text + BM25 with Faiss
Model 3	CLIP Image + CLIP Text + BM25 with Pinecone

Figure 14: Main models

Each of the above models in turn yields a new model obtained by adding SAM (and in particular the color change process) on top of it. Figure 15 below shows the models we used for the internal survey discussed in Section 4.1.

Model 1-Sa	Model 1 + SAM, same weights
Model 1-Sb	Model 2 + SAM, different weights (more semantic search)
Model 2	Model 2 + SAM, same weights
Model 3-Sa	Model 3 + SAM, same weights
Model 3-Sb	Model 3 + SAM, different weights (more semantic search)

Figure 15: Derivative models obtained from adding in the RGB color change and varying weights

Next, we describe the three basic models. Then, in Section 3.5, we describe how SAM was used on top of these basic models.

3.2 Model 1

This model relies on CLIP image and text search, which is described in more detail in Section 2.1. It creates an embedding for both the user's image and text and calculates the distance between the index embeddings from our database and each of our input's embeddings. This results in two distinct sets of distances, one for the image and one for the text. From here, we tried two different ways to combine these distances to get one set of results.

One way relies on finding the k nearest neighbors (kNN) for each of our input image and text embeddings and then intersecting the results to find common neighbors. We found that a good ordering of these common results is to base it on the text kNN ordering. Moreover, with this method, we found that having the model return a larger number of text neighbors compared to the image neighbors (about four times as many) yield improved results. Although this method worked well with some searches, the number of neighbors for both image and text had to be adjusted based on the search. For this reason, we ended up using an alternative method to combine distances that works better for any search.

This method consists of calculating an overall distance for each product in the dataset by combining the text and image distances and selecting the products with the smallest overall distance. We found that combining the scores through addition works well. Additionally, assigning more weight to the text distances yields overall better results. Specifically, we multiplied the text distances by 1.1. This preference for text embeddings stems from the fact that the original image alone may not precisely represent the desired outputs, making the textual description more important. This specific CLIP text and image combination became our Model 1.

3.3 Model 2

Building off of Model 1, we implement the BM25 search algorithm. As described earlier, given a text query, BM25 will assign each document a score. The higher the score the better. However, due to the fact that the BM25 scores are interpreted differently than the CLIP image and text distances, combining the search methods is a much more complicated task.

One way in which we combine these scores is to take the CLIP image distances and text distances and multiply them by negative 1 and then apply a softmax function. By doing so we transform the CLIP image and text results into scores so that higher scores translate to better results just like BM25. We then take these two new resulting vectors and the BM25 scores and multiply them all together element-wise.

The advantage of Model 2 is that when we compute scores in this way, no weights are required. We do not have to test to find the right linear combination of BM25 scores with CLIP distances. This way of combining the search results has worked well, but more experimenting and testing is required to find the best combination.

3.4 Model 3

This model, unlike all the other ones, uses Pinecone instead of Faiss to create a hybrid sparse-dense index.[3] Using the terminology introduced in Section 2.3.2, this model uses a single-stage retrieval system, as opposed to a two-stage retrieval and reranking system. This is different from the other models.

To generate dense vectors, we utilized the CLIP model to encode inputs and the image tensor file where CLIP has already been applied to our dataset. While this approach aligns with our other models, what makes it unique is the incorporation of sparse vectors, a hybrid sparse-dense index, and a distinct ranking process, facilitated by its one-stage retrieval system, as mentioned above.

Sparse vectors were generated with the Pinecone Text Client's implementation of BM25 [35].[4]. The process involves creating a BM25 object using the `BM25Encoder` class, then fitting the corpus (i.e., all titles) to the object, effectively training the BM25 model. The `fit` method in the class calculates the document frequency (DF) for each term in the corpus, iterating through the documents to calculate the term frequency (TF) and update the count of documents containing each token. It is also worth noting that `BM25Encoder` has a built-in NLTK-based tokenizer, which takes care of case-sensitivity, as well as removing punctuation marks and stop words. Once fitted, the encoder can encode new texts into sparse vectors using the BM25 algorithm.

The `encode_documents` method in the fitted BM25 model class takes either a single document or a list of documents as input and encodes them into sparse vectors using the fitted BM25 model. For a single document, it calculates the term frequency (TF), normalizes it using the BM25 formula, and returns the resulting sparse vector. When provided with a list of documents, it encodes each document individually and returns a list of corresponding sparse vectors.

We then perform the upserting process for creating a hybrid index in Pinecone [9]. This process handles data in batches to efficiently encode both metadata (which consists of the data from all columns

[3] Recall that sparse vectors are better suited for syntax-based search, whereas dense vectors are better suited for semantics-based search. For more discussion on sparse and dense vectors, see Section 2.3.2.

[4] We have also tried using SPLADE [5] from the Pinecone Text Client library [35] to produce sparse vectors. The results, however, were much worse than BM25. The poor results might have been the result of a mistake in the code, but we haven't been able to identify it. Additionally, it took a significant amount of time (around 3 hours) to create sparse vectors for all of our titles. In contrast, it took a couple of minutes to create sparse vectors using BM25.

except `id` and `year` and allows for filtering search results if necessary) and image features from the tensor into sparse BM25 vectors and dense CLIP vectors, respectively. Thanks to the use of the image tensor, this process is remarkably fast, taking less than two minutes[5]. Once the upserting process is completed, the hybrid index is ready for querying.

With the hybrid index, various types of searches are possible. We can perform image-only search (using dense-only vectors), text-only search (using sparse-only, dense-only, or both types of vectors), and combined image and text search. Note that despite not having dense vectors encoding the titles in our hybrid index,[6] we can still use dense-only vectors for text-only search, thanks to the CLIP model's parallel pre-training on pairs of images and texts. However, for image-based search, only dense vectors are used since images were only encoded into dense vectors. When performing a combined image and title search, we have the option to use either dense-only vectors or a combination of dense and sparse vectors. It is also possible to assign weights to the sparse and dense parts; see Section 2.3.2 for details.

As explained in Section 2.3.4, the query vector should contain both sparse and dense components. Let us describe the process of obtaining the dense parts of the queries. For text-only queries, we utilize the CLIP Tokenizer to generate a dense vector. On the other hand, if the query includes an image, we employ the CLIP Processor to convert it into a dense vector. Now, let's discuss how we handle text queries to create sparse vectors. We achieve this by using the `encode_queries` method of the fitted BM25 model. If a single query is provided, it calculates the term frequency for the query, computes the inverse document frequency (IDF) using the BM25 formula, and returns the corresponding sparse vector. In the case of a list of queries, each query is encoded individually, and a list of corresponding sparse vectors is returned. After the query vector is ready, the Pinecone performs similarity search as described in Section 2.3.4.

3.5 Adding RGB color change

For each of the three models above we also experimented with adding in the RGB color change described in Section 2.4.2.2. We compared adding in the RGB color change to Model 1 without changing any weights – this is Model 1-Sa – to a model where a heavier weight was assigned to the image distances to give more importance to semantic image similarity. This second model is what we called Model 1-Sb, where the image distances are weighted by 1.6. Similarly we investigated different values for alpha in association to Model 3 to adjust for how much the image query should influence the results. In this case we looked at increasing alpha from 0.3 to 0.5, which results in more semantic-based search. Note there was no simple way to modify weights for Model 2 and thus adding RGB color change resulted in only one additional model.

4 Model selection, evaluation, and discussion

In order to assess how well each model performs we decided to conduct two surveys – internal and external – to collect participant opinions. We focused on the ability of these models to perform an image search with a modification in color. In the internal survey, we were looking at the quality of the first 12 search results overall. In contrast, in the external survey, we were looking at the mean reciprocal rank statistic.

[5] If we had used images directly, it would have taken around 3 hours.

[6] The Pinecone hybrid index only supports one family of dense vectors. If we had used CLIP to generate title embeddings and stored them in the hybrid index, it would have occupied all available space, and we would have no room for the dense vectors generated from the images. As a result, we wouldn't have been able to perform image search.

4.1 Internal survey

The purpose of the internal survey was to narrow down the list of options, making it easier for external participants to complete a survey. Our selection process prioritized models whose top twelve results best represented the input searches which consisted of both an image of the desired product and a text query specifying the desired color.

As mentioned earlier, we have three basic models, or, more precisely, three infinite families of models, due to an infinite number of possible choices for weights. To narrow down the options, we compared the following models:

- Model 2 vs Model 3
- Model 2-S vs Model 3-Sa vs Model 3-Sb
- Model 1-Sa vs Model 1-Sb

The weights used in these models for the internal comparison, as well as in Model 1, are given in Figure 16.

Model	Weights
Model 1	1.1*text + image
Model 2	N/A
Model 3	alpha = 0.3
Model 1-Sa	1.1*text + image
Model 1-Sb	text + 1.6*image
Model 2-S	N/A
Model 3-Sa	alpha = 0.3
Model 3-Sb	alpha = 0.5

Figure 16: Different weights for each model

4.2 External survey

After this initial selection step, we sent out a revised survey to many people not on our team, including our boot camp cohort. The survey contained results from the four chosen models listed below and asked participants to pick the result they would click on if they were performing the search.

- Model 1
- Model 3
- Model 1-Sa
- Model 3-Sa

The weights used in these models are the same as in Figure 16. This allowed us to calculate the mean reciprocal rank (MRR) which is found using the following formula:

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i}$$

where Q is the number of queries and rank is the position of the highest-ranked answer. In our case, Q is nine and since we have multiple participants we average the MRR scores for each model over all participants.

Although we considered other selection and assessment tools that might have been more objective, we believe that the surveys allowed us to gather important insights from actual users which aligns with the objective of improving the user experience.

4.3 Survey results

From the first internal survey results, we selected Models 3, 1-Sa, and 3-Sa to be put into the external survey, and we complemented them with Model 1. A total number of 41 participants completed the external survey. The table below reports a summary of the MRR scores for each model. These were computed by first calculating an MRR score for each survey submission and model and then averaging over all submissions.

	Model 1	Model 3	Model 1-Sa	Model 3-Sa
Overall MRR	0.39	0.33	0.38	0.41
Avg Ranking	4.95	5.58	4.76	4.29

Figure 17: Rankings of models in the survey

Since a higher MRR score is better, Model 3-Sa performs best under this metric, closely followed by Models 1 and 1-Sa. In addition to the MRR, we calculated the average ranking for each model which are summarized above. Overall, participants selected between the 4th and 6th search result on average across the four models. Once again, Model 3-Sa performs best with an average ranking of just over 4.

Looking at the individual results makes it clear that Model 3 was struggling with the lack of semantic image similarity, in particular concerning color and shape. For example, when given an image of a red sock and the word “blue”, the majority of the results were navy blue socks, with only one bright blue result towards the end. The participants of the external survey dispreferred such dark blue results, partly because the corresponding images look closer to black. The other models did much better in this respect: in the mentioned example, they outputted many bright blue socks. The reasons for these differences are clear. Namely, Models 1 and 1-Sa perform a purely semantic search without using BM25 at all. Model 3-Sa outperformed Model 3 because the input image was a bright blue sock (after the segmentation with SAM and color change), leading to more accurate bright blue results.

The above suggests that the parameter alpha used in Model 3 might be too small, undervaluing the semantic input (currently 70% sparse BM25 vectors and 30% dense CLIP vectors). It would be interesting to explore how the results change by increasing alpha.

5 Conclusion

5.1 Summary

Through a variety of tools, we have been able to provide a method for a user to input both an image and a text query into their search. We primarily focused on a multimodal search where the user would input an image and a color as text to modify their search. Since each tool has its strengths and weaknesses, we explored which ones to include and with what weights, giving us potentially infinitely many models. In optimizing our models, we considered both recall and ranking, performing human judgment surveys to see how our models performed. We also implemented two versions of a web app through Streamlit that allows users to search with some of our best models. At this time, the limited version uses Model 2, while the full version uses a hybrid model with SAM color change.

5.2 Final recommendation

At this stage of the project, we believe that the models that incorporate more of these tools perform better. Thus, we recommend using a model similar to Model 3-Sa that combines SAM color change, CLIP image embeddings, CLIP text embeddings, and BM25. However, we truly believe that the Model 3-Sa can be improved given more time.

5.3 Future directions

While we are content with the tools that we are currently using for our search method, we would like to continue to explore the optimal way to combine the impact of each tool on the final search result. To do this, we would want to make adjustments to the way the tools are weighted in determining the search output. This would require further external surveys focusing more on different weights rather than different combinations of tools, as our initial survey did. In addition to adjusting these weights we would also like to overcome the issue of altering an image of a clothing item that starts out with a pattern as is discussed in Section 2.4. In a similar vein, our search method could benefit from a larger database of fabric swatches in order to make our search method all the more customizable. With an improved pattern search, we would want to resurvey to compare how the models perform with this feature, since the initial round of surveying solely focused on color change.

Our search method could also benefit from further work on optimizing the run time of the Segment Anything Model. Currently, in our Streamlit platform, it is time consuming to implement SAM. Therefore it is not always justifiable to include this addition to our search. Thus our current suggestion would be to make the use of Segment Anything an optional component for the user to apply in the situation where they are unhappy with their search results and therefore willing to wait longer in order to improve the search output. In addition to optimizing the run time of SAM in Streamlit, there are a few ways in which we would like to make the application more user friendly. We would like to combine both text boxes into one text query and extract the pattern and color information from that text instead of making those additional tasks for the user to input.

5.4 Acknowledgements

First of all, we want to thank our mentor Chris Miller for the invaluable guidance he provided. Additionally, we are grateful to eBay for providing us with this project. We also want to thank the University of Minnesota and the Institution for Mathematics and Its Applications for hosting the Math-to-Industry Bootcamp. Specifically, we appreciate Thomas Höft and Dan Spirn for organizing this bootcamp and offering much advice along the way. Code and grammar assistance was provided by ChatGPT.

References

[1] *Algolia - Comparing the best e-commerce search solutions*. URL: https://www.algolia.com/blog/ecommerce/comparing-the-best-ecommerce-search-solutions/.

[2] Arnon Dagan, Ido Guy, and Slava Novgorodov. "Shop by image: characterizing visual search in e-commerce". In: *Information Retrieval Journal* 26.1 (2023), p. 2.

[3] *Encord - Meta AI's New Breakthrough: Segment Anything Model (SAM) Explained*. URL: https://encord.com/blog/segment-anything-model-explained/ (visited on 07/28/2023).

[4] *Faiss Indexes on Facebook's GitHub Repository*. URL: https://github.com/facebookresearch/faiss/wiki/Faiss-indexes (visited on 07/28/2023).

[5] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. *SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking*. 2021. arXiv: 2107.05720 [cs.IR].

[6] *Hugging Face - CLIP*. URL: https://huggingface.co/docs/transformers/model_doc/clip (visited on 07/28/2023).

[7] *Hugging Face - Fine tuning CLIP with Remote Sensing (Satellite) images and captions*. URL: https://huggingface.co/blog/fine-tune-clip-rsicd (visited on 07/28/2023).

[8] *Hugging Face - Transformers*. URL: https://huggingface.co/docs/transformers/index (visited on 07/28/2023).

[9] *Hybrid Search for E-Commerce with Pinecone*. URL: https://docs.pinecone.io/docs/ecommerce-search (visited on 07/28/2023).

[10] *Image of Black Dress*. URL: https://freepngimg.com/png/31377-dress-free-download.

[11] *Image of Black Purse*. URL: https://freepngimg.com/png/7459-women-bag-png-image.

[12] *Image of Blue Backpack*. URL: https://freepngimg.com/png/9810-backpack-png-hd.

[13] *Image of Blue Purse*. URL: https://freepngimg.com/png/144384-blue-bag-ladies-free-download-image.

[14] *Image of Blue Shirt*. URL: https://freepngimg.com/png/2100-blue-dress-shirt-png-image.

[15] *Image of Men in Blue Shirt and Red Shorts*. URL: https://www.crewclothing.co.uk/.

[16] *Image of Men's Sunwashed Canvas Shirt*. URL: https://www.llbean.com/llb/shop/506874?attrValue_0=Slate&gclsrc=aw.ds&originalProduct=68855&pla1=0&productId=1150401&qs=3155278&sku=1000005402.

[17] *Image of Men's Sunwashed Canvas Shirt*. URL: https://www.jiffyshirts.com/bellacanvas-B6400.html?ac=Orchid.

[18] *Image of Model in Black Dress*. URL: https://candieanderson.com/.

[19] *Image of Red Jacket*. URL: https://freepngimg.com/png/150672-leather-jacket-red-free-download-image.

[20] *Image of Red Purse*. URL: https://freepngimg.com/png/7461-red-women-bag-png-image.

[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs". In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547.

[22] *Kaggle - Fashion Product Images Dataset*. URL: https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset (visited on 07/11/2023).

[23] Alexander Kirillov et al. "Segment Anything". In: *arXiv:2304.02643* (2023).

[24] *LearnOpenCV - Segment Anything – A Foundation Model for Image Segmentation*. URL: https://learnopencv.com/segment-anything/ (visited on 07/28/2023).

[25] *MathWorks - ResNet50*. URL: https://www.mathworks.com/help/deeplearning/ref/resnet50.html (visited on 07/28/2023).

[26] *Meta AI - Introducing Segment Anything: Working toward the first foundation model for image segmentation*. URL: https://ai.meta.com/blog/segment-anything-foundation-model-image-segmentation/ (visited on 07/28/2023).

[27] *Pinecone - CLIP*. URL: https://openai.com/research/clip (visited on 07/28/2023).

[28] *Pinecone - Dense Vectors: Capturing Meaning with Code*. URL: https://www.pinecone.io/learn/series/nlp/dense-vector-embeddings-nlp/ (visited on 07/28/2023).

[29] *Pinecone - Multi-modal ML with OpenAI's CLIP*. URL: https://www.pinecone.io/learn/series/image-search/clip/ (visited on 07/28/2023).

[30] *Pinecone - Nearest Neighbor Indexes for Similarity Search*. URL: https://www.pinecone.io/learn/series/faiss/vector-indexes/ (visited on 07/28/2023).

[31] *Pinecone - Sparse-dense embeddings*. URL: https://docs.pinecone.io/docs/hybrid-search (visited on 07/28/2023).

[32] *Pinecone - SPLADE for Sparse Vector Search Explained*. URL: https://www.pinecone.io/learn/splade/ (visited on 07/28/2023).

[33] *Pinecone - Text-to-Image and Image-to-Image Search Using CLIP*. URL: https://www.pinecone.io/learn/clip-image-search/ (visited on 07/28/2023).

[34] *Pinecone - What is a Vector Database?* URL: https://www.pinecone.io/learn/vector-database/ (visited on 07/28/2023).

[35] *Pinecone Text Client GitHub Repository*. URL: https://github.com/pinecone-io/pinecone-text (visited on 07/28/2023).

[36] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[37] *Segment Anything Website*. URL: https://segment-anything.com/ (visited on 07/28/2023).

[38] *Segment Anything's GitHub repository*. URL: https://github.com/facebookresearch/segment-anything/tree/main (visited on 07/28/2023).

[39] *Startup Bonsai - 25+ Top eBay Statistics 2023 (User And Revenue Data)*. URL: https://startupbonsai.com/ebay-statistics/#:~:text=There%20are%20around%201.6%20billion%20live%20listings%20on%20eBay%20today,devices%20as%20of%20Q1%202022.

[40] *Statista - Retail e-commerce sales worldwide from 2014 to 2026*. URL: https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/.

[41] *Streamlit Discussion Page on git-lfs*. URL: https://discuss.streamlit.io/tag/git-lfs (visited on 07/28/2023).

[42] *This project's GitHub repository*. URL: https://github.com/JeremyShahan/VisualSearch.

[43] *Transformers - Deep Learning Notes by Subir Varma Sanjiv Ranjan Das*. URL: https://srdas.github.io/DLBook2/Transformers.html (visited on 07/28/2023).