Vectorized Image Search with CLIP and Faiss

Author(s)
Jihong Huang, Jiachen Ma, Liping Yin, Rongxiang Zhi, Lidian Zhuo
kolbehuang@ucla.edu, stephenmaaa@ucla.edu, lipyin@ucla.edu,
zhirongxiang@ucla.edu, zld3794955@ucla.edu

Abstract

Many studies have been carried out to integrate multi-modal data into a global feature space. In such a dataset, heterogeneous data like text, images, and videos, could be accessed and processed in a uniform manner. However, the integration of multi-modal data also means the loss of information, which makes it necessary to find methods that can extract relevant information from the global dataset both effectively and efficiently. That is, the search results from the dataset should have good quality and can be obtained at a low time cost. In this project, we would like to compare both search quality and efficiency of several search methods in a dataset uniformly storing embedded caption-image pairs. Specifically, we used CLIP to pre-process the dataset into high-dimensional vectors. Then, we applied different search methods, such as Nearest Neighbors and various Faiss methods with different parameters, on text-to-image and image-to-image search. Finally, we utilized precision@k and NDCG as the metrics for measurement. The text or image to search for might not only be selected from the dataset but also could be arbitrarily generated. During our evaluation, we discovered the trade-off between search quality and efficiency. As a result, we found that the clustering Faiss built on inner product could reach the optimal balance.

1 Introduction

1.1 Motivation

In 2021, OpenAI introduced CLIP which could integrate multi-modal data into a global feature space, and those heterogeneous data such as texts and images were stored as embedding vectors in a uniform manner. This model can facilitate many downstream Machine Learning and Data Analysis tasks. However, the integration of multi-modal data also brings the loss of information, which requires methods that can extract relevant information from the dataset both efficiently and effectively to best utilize the model. Thus, search results should be retrieved at a good quality and low time cost. In this project, we would like to select several search methods, including the plain kNN and multiple variants of Faiss, and then compare them in terms of both search quality and efficiency.

1.2 Dataset

The dataset used for evaluating similarity search is MS COCO (Microsoft Common Objects in Context) 2017 Dataset. MS COCO 2017 dataset is a large-scale high-quality crowd-labeled dataset. For each image, there are 1 to 5 captions. The training set contains 118K images and the validation set contains 5,000 images.

1.3 Vector Transformation (CLIP)

To transform text-based and/or image-based data into vectors, we used the Contrastive Language-Image Pre-Training neural network (CLIP)[5] as the pre-trained base model. The CLIP model was trained on various caption-image pairs (400 million in total and each class includes up to 20,000 pairs). The specific pre-trained CLIP model loaded is Vision Transformer ViT-B/32. The CLIP model is composed of a text encoder and an image encoder (Figure 1). By embedding the original text and image, CLIP can compute the cosine similarity between the given pairs.

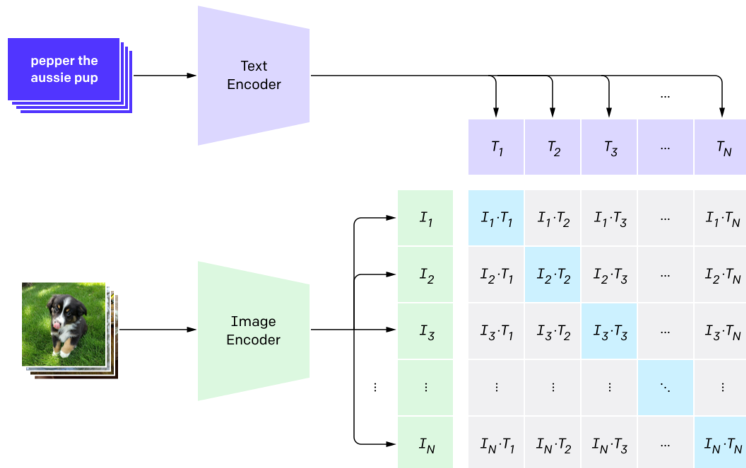


Figure 1: Transforming texts and images into same vector space

1.4 Finetuning

In order to generate more informative vector embedding for text and image, we fine-tuned the pre-trained ViT-B/32 on MS COCO 2017 dataset. For the COCO training set, the model is trained on 118K caption-image pairs. We used the same training method used in CLIP: in every epoch, for each batch of caption-image pair inputs, CLIP learns to extract features of each modality by training the text encoder and image encoder jointly, then CLIP computes the pairwise cosine similarities by taking the dot product of text embedding and image embedding, and the final loss function l is the average of the cross-entropy loss from text encoder and the cross-entropy loss from image encoder. For hyperparameter choice, we used a smaller learning rate of 1e-5 for fine-tuning; we chose Adam optimizer using betas in (0.9, 0.98), epsilon in 1e-6 and weight decay in 0.2, which was the same hyperparameters used in CLIP. We used a batch size of 64 to fine-tuned the model for 10 epochs where each took around 40 mins. The COCO 2017 Validation Set was used to evaluate the performance.

2 Search Methods

After mapping texts and images into the same vector space, we started to build our search functions. The question is how to store these vectors (the "data structure") and how to find the best-matched results given a query (the "search method"). The easiest and most apparent way is using the old-school k-Nearest-Neighbor (kNN) where we simply store all the vectors. Upon receiving an input query, we simply find its nearest neighbors from the storage and return them as the results. However, while it guarantees the best-matched results based on distance, we may have to compromise the efficiency and cost, such as query time and RAM usage. We may even have to sacrifice some accuracy in exchange for some efficiency boost, especially when we have a large amount of data to search on. Thus, there are various ways to achieve or emulate the kNN search. In this project, we mainly focused on Faiss.

Facebook AI Similarity Search (Faiss)[3] is a library designed for efficient similarity search on large amounts of vectors. It contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM. Given a set of vectors x_i in dimension d, Faiss builds a data structure called "index" in RAM. The storage operation of vector x_i is achieved by calling the $add()$ method on the index. After the final structure is completed, given a query vector x in dimension d, it efficiently performs the operation:

$$i = \underset{j}{argmin}||x - x_j||$$

where $||.||$ could be either Euclidean (L2) distance or dot-product distance.

In other words, solving the $argmin$ problem is equivalent to performing the $search(x)$ operation on the index. Of course, Faiss is a very flexible library as it provides many parameters for us to tune on. For instance, there are many ways to construct the index, measure the similarity and so on, where each combination comes with different preferences over speed, accuracy, and RAM usage. For this project, we are interested in investigating three different variants of Faiss: the "vanilla" method, the "clustering" method and AutoFaiss.

2.1 Faiss (Vanilla)

This refers to the "flat" indexes in Faiss [2]. Flat indexes simply encode the N vectors into codes of a fixed size d and store them in an array of size $N \times d$. At search time, all the stored vectors are decoded sequentially from the array and compared to the query vector. Thus, it has a theoretical search runtime of $O(N)$, though some small tricks (such as compression) can still be played to slightly boost the speed. It is the closest emulation of kNN since it covers the entire search space and guarantees to return the most similar results.

2.2 Faiss (Clustering)

This refers to the "IndexIVF" indexes in Faiss. We use a partition-based method based on multi-probing. The feature space is first partitioned into possibly uneven C "cells". All vectors are assigned to one of these cells by a quantization function. A typical function is k-means where each data point is assigned to the closest centroid. At query time, a set of $nprobe$ cells which are assumed to be top relevant to the query input is selected ($nprobe$ is an adjustable parameter), and the query is compared to each of the vectors inside these selected cells. Thus, only approximately a $nprobe/C$ fraction of the original search space is compared to the query, which reduced the query time. However, we are no longer guaranteed to find the best result because a failure appears when the cell of the nearest neighbor(s) of a given query is not selected.

2.3 AutoFaiss

One interesting library we found is called AutoFaiss[1], which provides a wrapper for Faiss, but saves hassle by automatically creating the "best" Faiss indexes with the "most optimal" parameters. Upon calling, it enumerates over multiple Faiss indexes on the same set of vectors, and returns the best indexing parameters to achieve the highest recalls given memory and query speed constraints. As it sounds too good to be true, we remain skeptical about their claims and decide to include it in our evaluation.

2.4 sklearn.neighbors.NearestNeighbors

This is a kNN implementation provided by the sklearn library. The principle behind k nearest neighbor[4] methods is to find the top k training samples closest in the distance to the new point and predict the label from these k samples. The distance we use for metric measurement is cosine distance.

3 Evaluation

3.1 Measurements

To measure the search results from different search methods on different tasks, we used precision@k and Normalized Discounted Cumulative Gain (NDCG) as the criteria. Depending on the existence of ground truth corresponding to the inputs within the domain of our tasks, the measurements are divided into two types – internal measurements and external measurements.

3.1.1 Internal Measurements

Suppose that a given input, which can be either a caption or an image, belongs to a caption-image pair in the dataset. We call such an input "internal", meaning that the ground truth of this inquiry exists in the dataset. If the input is a caption, then the ground truth of the task searching for an image based on the input caption will be its corresponding image; if the input is an image, then the ground truth of the task searching for an image based on the input image will be the input itself. In this case, we use precision@k as the internal measurement.

Let's denote the input X and the corresponding ground truth Y. Given the input X, any search method will return its top k results $\{X_1, \ldots, X_k\}$. If $\exists i \in \{1, \ldots, k\}$ s.t. $X_i = Y$, we assume that this search method successfully finds the ground truth of the input. Otherwise, the search method fails on the input X. Suppose we evaluate N examples, then the internal precision@k is the fraction of successful results returned by a search method with top k,

$$prec@k = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\exists j \in \{1, \ldots, k\} \textbf{ s.t. } X_j^{(i)} = Y^{(i)}\right]$$

where $X_j^{(i)}$ is the j-th search result given input $X^{(i)}$ and $Y^{(i)}$ is the ground truth of input $X^{(i)}$.

We did not use Normalized Discounted Cumulative Gain (NDCG) as a measurement for an internal input. Because there exists exactly one ground truth in the dataset, the relevance is 1 if and only if the result is the same as the ground truth. This means that among the top k search results there is at most one that is relevant, which is not very informative. In other words, as the full true relevance is not available, the Ideal Discounted Cumulative Gain (IDCG) does not exist, which makes NDCG inappropriate here.

3.1.2 External Measurements

Suppose that a given input, which can be either a caption or an image, does not exist in any caption-image pair in the dataset. We call such an input "external". Since a well-defined ground truth in the dataset is not available, we define the ground truth to be the closest result found by the plain k-nearest-neighbors (kNN) using the metrics of cosine similarity. If the input is a caption, the ground truth will be the image of the highest cosine similarity with the input caption in terms of embedding vectors given by the plain kNN; if the input is an image, the ground truth will be the image that has the highest cosine similarity to the input image, given by the plain kNN. In this case, we use both precision@k and NDCG as external measurements.

Let's denote the input X and the corresponding ground truth Y. Given the input X, any search method will return its top k results $\{X_1, \ldots, X_k\}$. If $\exists i \in \{1, \ldots, k\}$ s.t. $X_i = Y$, we assume that this search method successfully finds the ground truth of the input. Otherwise, the search method fails on the input X. Suppose we evaluate N examples, the external precision@k is the fraction of successful results returned by a search method with top k,

$$prec@k = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\exists j \in \{1, \ldots, k\} \textbf{ s.t. } X_j^{(i)} = Y^{(i)}\right]$$

where $X_j^{(i)}$ is the j-th search result given input $X^{(i)}$ and $Y^{(i)}$ is the ground truth of input $X^{(i)}$.

Assume the same setup of input X and ground truth Y, and the search method returns its top k results $\{X_1, \cdots, X_k\}$. Given the same X, the plain kNN also returns its top k results $\{Y_1, \ldots Y_k\}$, where Y_1 is exactly the ground truth Y. Consider any search result X_i, we represent its relevance to Y using the cosine similarity between X_i and Y in terms of their embedding vectors:

$$rel(X_i) = CosineSimilarity\Big(v(X_i), v(Y)\Big) = \frac{\langle v(X_i), v(Y)\rangle}{||v(X_i)|| \cdot ||v(Y)||}$$

where $v(X_i)$ represents the embedding of X_i and $v(Y)$ represents the embedding of Y.

Then, we can compute the Discounted Cumulative Gain @k (DCG$_k$) of the search results $\{X_1, \cdots, X_k\}$ with respect to the ground truth Y:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel(X_i)} - 1}{\log_2 (i+1)}$$

Since we define plain kNN as the ground-truth method, we take the top k results from kNN to compute the true relevance and Ideal Discounted Cumulative Gain @k (IDCG$_k$) of the true results $\{Y_1, \cdots, Y_k\}$ with respect to the ground truth Y:

$$rel(Y_i) = CosineSimilarity\Big(v(Y_i), v(Y)\Big)$$

$$IDCG_k = \sum_{i=1}^{k} \frac{2^{rel(Y_i)} - 1}{\log_2 (i+1)}$$

Finally, the Normalized Discounted Cumulative Gain @k (NDCG$_k$) is computed:

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

The NDCG score with range [0,1] can measure the quality of the top k results of each search method compared with the plain kNN. A higher NDCG score on an external input means that a search method is closer to the quality of the plain kNN, and vice versa. Thus, theoretically we are measuring the relative quality. Furthermore, we observed that though the plain kNN had a satisfying search quality in our domain of task, its efficiency was much lower than any other search methods. This leads to the discussion of the trade-off between search quality and efficiency below.

3.2 Results

3.2.1 Internal Task: Captions to Images

Obviously, the search method with the best quality was kNN, as we could see in Figure 2(a) that the sklearn kNN had the best precision@k for all $k = 1, 3, 5, 7$. The best methods following the kNN were Faiss with clusters built on inner product, Faiss without clustering built on inner product, and AutoFaiss. These three methods still have a competitive quality because their precision@k's were at most 10% lower than those of kNN.

However, Figure 2(b) illustrated that it took the sklearn kNN over 30 ms to execute a single query. AutoFaiss took around 10 ms, and all the other Faiss methods took around 3 ms. Even though the Faiss methods took more time to initialize the indices (Table 1), it might be still worthwhile as the time cost of performing a single query by Faiss with clustering is much lower than all the other methods.

Therefore, there exists a trade-off between search quality measured by precision@k and search efficiency measured by the runtime on a single query. The sklearn kNN gave the best precision@k but also the highest time cost. Thus, the search method reaching a balanced point between quality and efficiency should be Faiss with clustering built on inner product.

Table 1: Runtime of a single internal query (caption)

Search Method	Initialization (ms)	Runtime of internal caption (ms)
Plain kNN	10.925	30.993
Faiss, Vanilla, L2 Distance	2.237	3.667
Faiss, Vanilla, Inner Product	2.067	3.594
Faiss, Clustering, L2 Distance	133.556	2.504
Faiss, Clustering, Inner Product	128.749	2.463
AutoFaiss	16886.296	10.145

3.2.2 Internal Task: Images to Images

The search quality of different search methods was almost equally good since all of the precision@k's were close or equal to 1 (Figure 3(a)).

The runtime of a single query maintained the same trend as in the previous task (Figure 3(b)).The sklearn kNN took the longest time, followed by AutoFaiss and all the other Faiss methods. Faiss with clustering built on Euclidean distance took the least single-query runtime. Therefore, since the search quality measured by precision@k were saturated in this task, we found that Faiss with clustering built on Euclidean distance should be the method that reached the well-balanced point between quality and efficiency.

Table 2: Runtime of a single internal query (image)

Search Method	Initialization (ms)	Runtime of internal image (ms)
Plain kNN	10.925	33.912
Faiss, Vanilla, L2 Distance	2.237	3.592
Faiss, Vanilla, Inner Product	2.067	3.592
Faiss, Clustering, L2 Distance	133.556	0.172
Faiss, Clustering, Inner Product	128.749	2.515
AutoFaiss	16886.296	10.337

3.2.3 External Task: Captions to Images

Recall that we used the closest image and top k results returned by the plain kNN as the ground truth and the true relevance respectively. Figure 4(a) indicated that none of the Faiss methods could find the ground truth in the first search, but all of them succeeded within the top 3 results (except for Faiss with clustering built on Euclidean distance which failed in all cases). Such a failure was possible because the usage of clustering implied that we would overlook some of the clusters which might contain better results, and using Euclidean distance as the metric made it less able to capture the results with smaller cosine similarity.

Figure 4(b) showed that the NDCG score of each Faiss method was around 90% considering top 3 or more (except for Faiss with clustering built on Euclidean distance). Based on the definition of NDCG scores, they showed a search quality that was 90% of the plain kNN considering the top 3 or more results. If we consider the fact that the runtime of a single query using plain kNN (Table 13) was much longer, such a 10% cost of quality may be worthwhile for the great reduction in runtime.

The runtime of a single-query on an external caption were given in Figure 4(c). AutoFaiss took the longest time, followed by Faiss without and with clustering. Faiss with clustering built on Euclidean distance took the least runtime.

Overall, Faiss with clustering built on inner product had the best precision@k and NDCG$_k$, and second-lowest runtime, making it the optimal method for this task.

Table 3: Runtime of a single external query (text)

Search Method	Initialization (ms)	Runtime of internal image (ms)
Faiss, Vanilla, L2 Distance	2.237	3.565
Faiss, Vanilla, Inner Product	2.067	3.580
Faiss, Clustering, L2 Distance	133.556	0.135
Faiss, Clustering, Inner Product	128.749	2.338
AutoFaiss	16886.296	9.942

3.2.4 External Task: Images to Images

We could see in Figure 5(a) that all the Faiss methods gave similar precision@k, though Faiss with clustering built on Euclidean distance had the lowest.

In terms of NDCG scores of the Faiss methods, Figure 5(b) indicated that all the Faiss methods could reach an NDCG over 0.95 for top $k = 5, 7$, which meant that they could achieve over 95% quality of the plain kNN considering the top 5 or 7 results. If we consider the fact that the runtime of a single-query using the plain kNN (Table 24) was much longer, such a 5% cost of quality may be worthwhile for the great reduction in runtime.

The runtime of a single-query on an external image were given in Figure 5(c). AutoFaiss took the longest single-query time, followed by Faiss without and with clustering. Faiss with clustering built on Euclidean distance took the least runtime.

Overall, Faiss with clustering built on inner product had the best precision@k, NDCG$_k$, and second-lowest runtime, making it the optimal method for this task.

Table 4: Runtime of a single external query (text)

Search Method	Initialization (ms)	Runtime of internal image (ms)
Faiss, Vanilla, L2 Distance	2.237	3.592
Faiss, Vanilla, Inner Product	2.067	3.593
Faiss, Clustering, L2 Distance	133.556	0.171
Faiss, Clustering, Inner Product	128.749	2.522
AutoFaiss	16886.296	10.335

3.3 Complementary Analysis

3.3.1 Internal Task (Captions to Images)

It could be seen that the search results from "internal" captions were not satisfying since the precision@k's of all the search methods were below 0.7. One possible reason behind could be that the embedding model CLIP could not well encode a caption/image pair into embedding vectors that are sufficiently close in terms of cosine similarity. To estimate the baseline truth, we ran several additional experiments and found that the average cosine similarity between the embedding vectors of caption-image pairs was only 0.3059, and the maximum cosine similarity was only 0.4023. Thus, the distance between the embedding vectors of a caption-image pair is not inherently small. This fact makes it extremely difficult to find the ground truth as the relationship between the true caption-image pair is not strong enough.

The same problem did not happen in the case of searching on "external" captions because we used the plain kNN instead of simulating the "ground truth", which skipped the issue of low cosine similarity between any true pair of caption and image.

3.3.2 Faiss with Clusterting Built on Inner Product

Faiss with clustering built on inner product was observed to be the optimal method in all the tasks, possibly except for the one searching on internal images where precision@k was saturated. It used inner product, which was basically unnormalized cosine similarity, as its metric. Thus, it could well catch the results with desired properties. Besides, the application of clusters (a.k.a. partitions) enabled Faiss to only search through those clusters which it thought to be highly relevant to the query, saving runtime to give fairly good results. Therefore, this Faiss method should be satisfying in both quality and efficiency.

3.3.3 AutoFaiss

We mentioned that AutoFaiss was a Faiss method whose parameters were automatically optimized, which seemed that it should produce great performance. However, our results above indicated that its performance was not that outstanding as expected. It took an extremely long time to initialize, and its single-query runtime was still longer than the other "simpler" Faiss methods. One explanation for this counter-intuitive phenomenon was that our test dataset was too small for these Faiss methods, so the advantage of parameter optimization of AutoFaiss could not be well utilized. Besides, it took a long time to optimize its parameters, which might not be worthwhile for our tasks and small dataset, but very useful for those larger datasets. However, it was able to produce reasonable results without the hassle for us to manually select the appropriate parameters, unlike other "simpler" methods where we needed to tune the number of clusters/probes, metric types and so on.

3.4 Discussion

3.4.1 Limitations

Despite our efforts, there were still some limitations in our project. First, due to the limit of available resources, we failed to build and test on a vectorized dataset of caption-image pairs whose size is as large as tens of thousands or hundreds of thousands which those advanced Faiss methods were designed for. Therefore, we could not observe a more precise and apparent difference between these methods. Second, the embedding model CLIP did not well encode a pair of caption and image into a pair of similar embedding vectors, which made the task of searching on internal captions essentially challenging. Such a flaw blocked the full exploration of the search procedure from captions to images. Third, we used cosine similarity throughout the project as the criterion evaluating the similarity between two embedded images. However, in practice two different images may both well match the same caption/image though their cosine similarity is low due to various reasons. It would be better if a criterion that could measure the semantic meaning of an input was available (for example, it could be part of a Generative Adversarial Network).

3.4.2 Further Studies

As further research in the field of vectorizing multi-modal data, more similarity criteria should be investigated to better capture the semantic meaning of any given input. Also, more search methods, including more advanced variants of Faiss methods, are to explore to guarantee satisfying search quality while maintaining the necessary (or even real-time) efficiency. With further studies on these components, our project could be transformed to serve as a real-time caption generator given an image, a piece of audio or even a video.

4 Conclusion

We have investigated and evaluated the quality and efficiency of several search methods, based on a set of caption-image pairs embedded by CLIP: plain kNN, vanilla Faiss built on Euclidean distance, vanilla Faiss built on inner product, clustering Faiss built on Euclidean distance, clustering Faiss built on inner product, and AutoFaiss. During the experiments, we found that there exists a trade-off between search quality (represented by the goodness of search results) and search efficiency (measured by the runtime of a single query). The plain kNN constantly gave the best search results but also took the longest time to execute a single query, while the clustering Faiss built on Euclidean distance consumed the least time but gave relatively unsatisfying search results. Thus, we conclude that, based on our experiment settings, the clustering Faiss built on inner product could reach a well balance between search quality and efficiency.

References

[1] Criteo. (2022, September 7). Criteo/autofaiss. GitHub. Retrieved from `https://github.com/criteo/autofaiss`

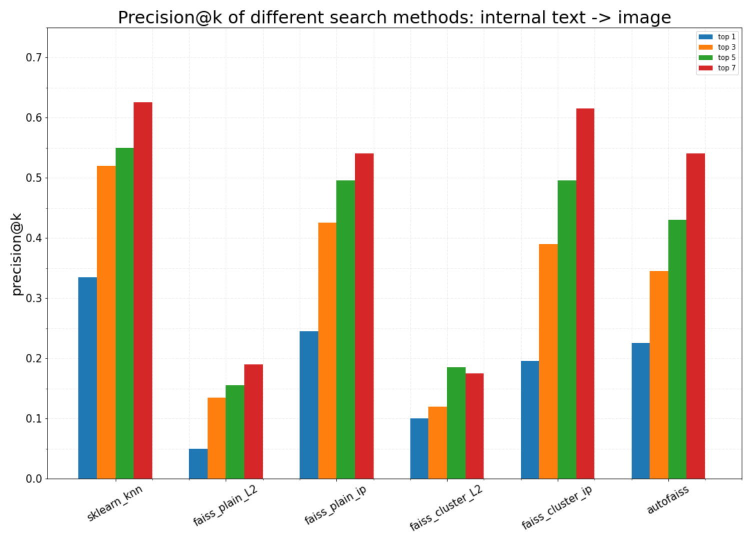
[2] Douze, M. (2022, April 27). Faiss indexes · facebookresearch/Faiss Wiki. GitHub. Retrieved December 12, 2022, from `https://github.com/facebookresearch/faiss/wiki/Faiss-indexes`

[3] Douze, M. (2022, March 11). Home · facebookresearch/Faiss Wiki. GitHub. Retrieved from `https://github.com/facebookresearch/faiss/wiki`

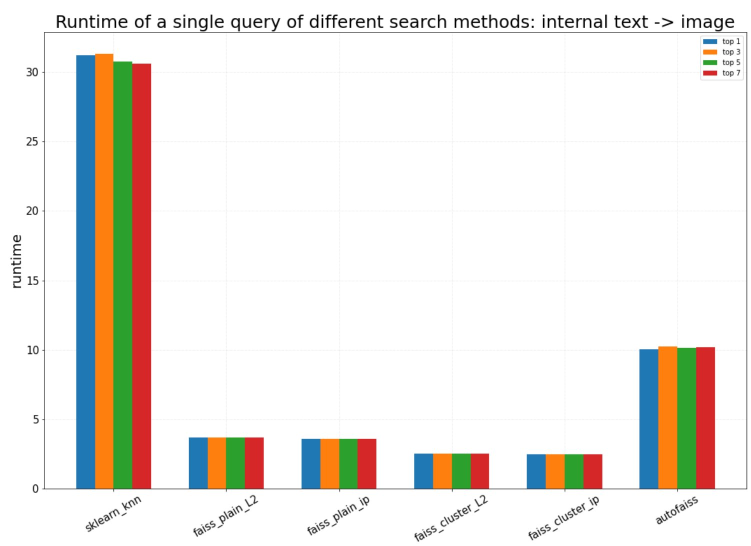
[4] nearest neighbors. scikit. (n.d.). Retrieved from `https://scikit-learn.org/0.15/modules/neighbors.html#neighbors`

[5] OpenAI. (2021, January 5). Clip: Connecting text and images. OpenAI. Retrieved from `https://openai.com/blog/clip/`

A Appendix

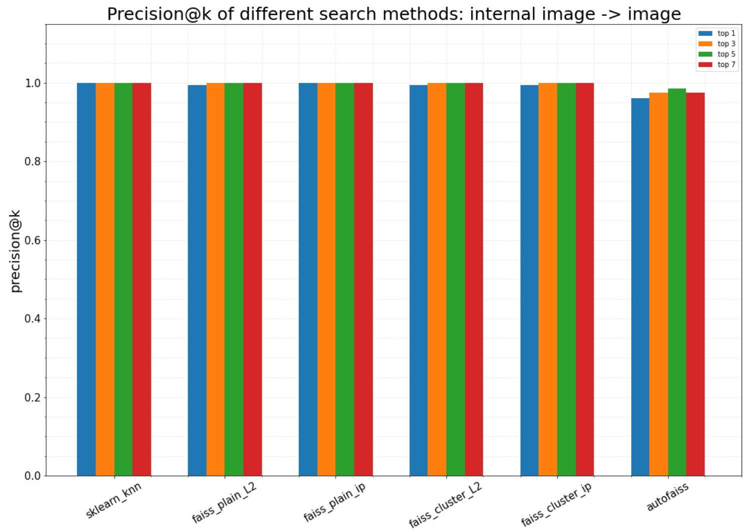


(a) precision@k of different methods

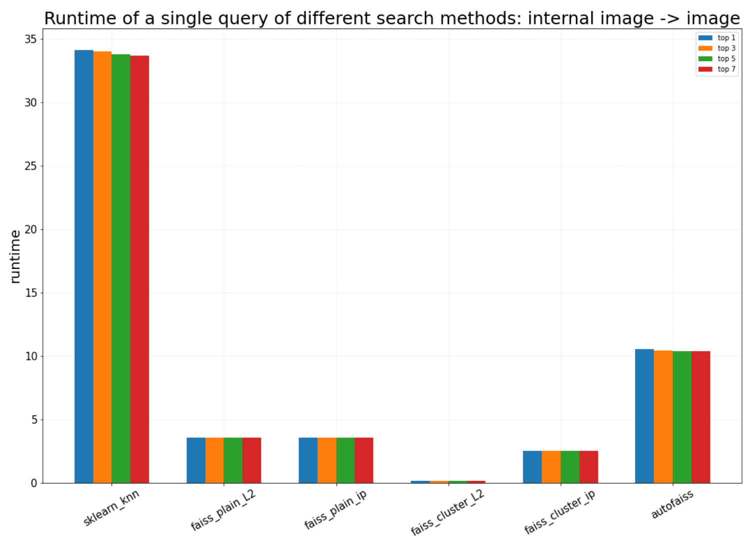


(b) single-query runtime of different methods

Figure 2: Text2Image results by internal measurements on different search methods

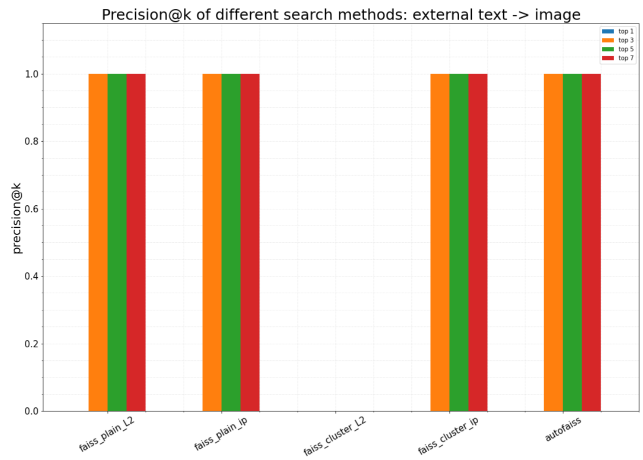


(a) precision@k of different methods

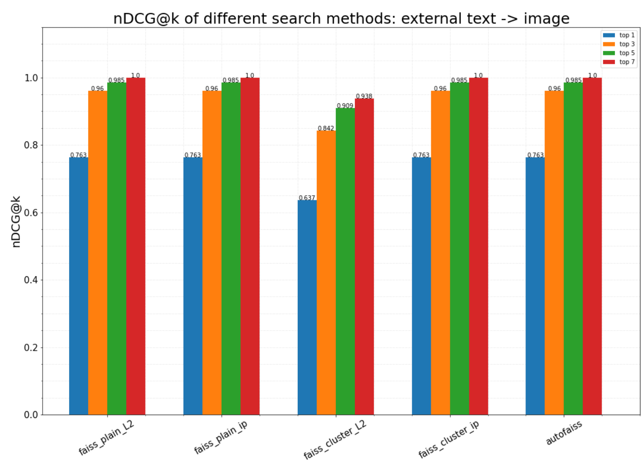


(b) single-query runtime of different methods

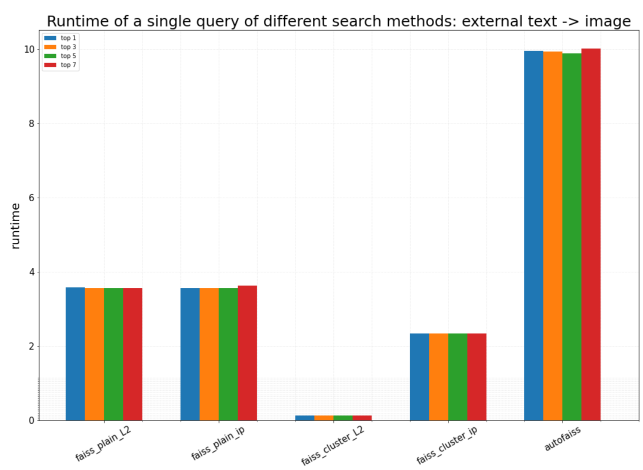
Figure 3: Image2Image results by internal measurements on different search methods



(a) precision@k of different methods

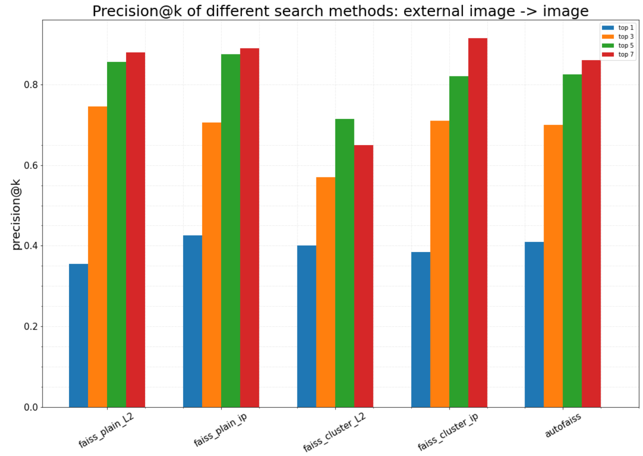


(b) NDCG of different methods

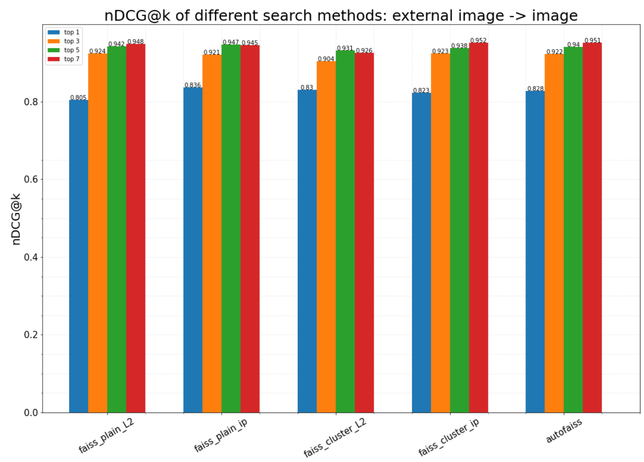


(c) single-query runtime of different methods

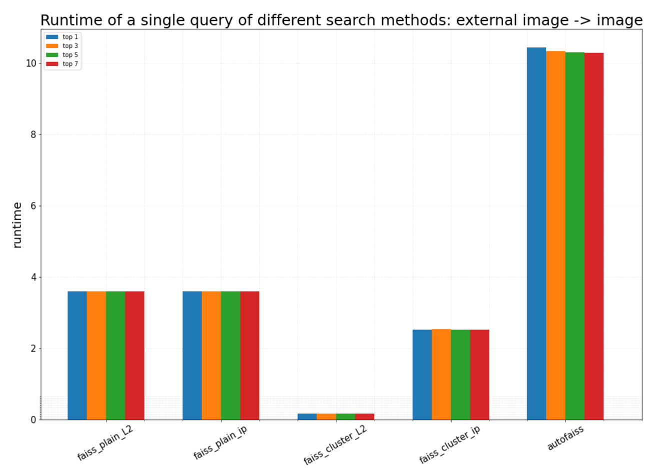
Figure 4: Text2Image results by external measurements on different search methods



(a) precision@k of different methods



(b) NDCG of different methods



(c) single-query runtime of different methods

Figure 5: Image2Image results by external measurements on different search methods