Vectorized Similarity Search in Multi-modal Databases

Hang Zhang **Yasmin Zhang** **Jiayueran Sun** **Wei Long** **Zhengwei Zhou**

Abstract

In this paper, we propose a novel approach to conducting similarity search for data with multiple modalities. We present a pipeline for this process including index initialization and similarity search. We build and compare the efficiency and effectiveness between kNN exact search and ANN methods such as HNSW and IVF. Furthermore, detailed comparison between multi-modal and single-modal data inputs shows the strength of multi-modal data sources in terms of both time and space. Our code for the experiments can be found on `https://github.com/YasminZhang/UCLACS260-Vectorized-Similarity-Search-in-Multi-modal-Databases`.

1 Introduction

Under real world situations, data from experiments and observations are recorded in many different formats, such as text, figure, medical scans, audio recordings, etc. This diversity result in the difficulty for downstream data analytics and machine learning tasks to be realized. The need to integrate and homogeneously represent data of different formats, scales, and distributions has led to the study of multi-modal data and multi-modal databases where such data can be stored, queried, and indexed.

In this paper, we choose similarity search as our main downstream task to accomplish. The k-nearest neighbor search(kNN) is a well-known mathematical way of similarity search. (Malkov et al. [2014]). However, such exhaustive search method is computational heavy. We try to use approximate k-nearest neighbor search to solve this problem and compare their performance. In our experiment, we build several search algorithms and compared their effectiveness and efficiency under different vector distance settings. We also dig into the difference between multi-modal query inputs and single-modal query inputs and generate valuable findings.

In section 2, we provide literature reviews to help solidify definitions in our project. In section 3, we show the detailed implementation of the pipeline. In section 4, we introduce the settings of our experiment and work through the detailed process of actual experiment, metrics design, and result evaluation.

2 Literature Review

2.1 Multimodal deep learning

As many machine learning algorithms are struggling to process large amounts of data, a technique named **vector embedding** is introduced to solve such problems. Vector embeddings are low-dimensional spaces that are translated from high-dimensional vectors. They help simplify machine learning by clustering semantically related inputs together in the embedding space (Weston et al. [2008]). In order to get embedding, we apply **feature engineering**, which is the process of extracting features from raw data using domain knowledge (Heaton [2016]). Because there are many sources of

data, **multimodal deep learning** is widely used to integrate the various data sources. It involves the blending of multiple data modalities, such as text and image data, which have different statistical characteristics. Multimodal deep learning requires specific modeling approaches and algorithms to effectively combine the different modalities (Ngiam et al. [2011]).

The state-of-the-art of multimodal deep learning is **Contrastive Language-Image Pre-training (CLIP)** (Radford et al. [2021]). CLIP relies on a substantial corpus of research on zero-shot transfer, natural language supervision, and multimodal learning. Through a machine learning technique called contrastive learning, an image encoder (e.g. ResNet50 (He et al. [2015])) and a text encoder (e.g. BERT (Devlin et al. [2018])) are jointly trained in CLIP to predict the ideal pairing of a batch of images and text. This method is different from traditional image challenges, which often require the model to choose one class from a huge number of classes (e.g. ImageNet (Deng et al. [2009])). This means it can embed the text and images into joint semantic space which allows us to use it for the most similar image for a given text or image. As a result, we can use it to find the most similar picture for a given text or image by embedding the text and images into a single semantic space.

2.2 Vector Similarity Search

Similarity search is one of the most popular uses of vector embeddings. Search algorithms like kNN and ANN require us to calculate the distance between vectors to determine similarity.

The **K nearest neighbors technique (kNN)** can be used to find the closest vectors in a space for a given query vector (Cover and Hart [1967]). The number of nearest neighbors is indicated by the hyperparameter k. A significant disadvantage of kNN is that the distance between each vector needs to be calculated in order to identify the closest vectors. If there are millions of vectors, this will be extremely inefficient.

To reduce the computation complexity added by an exhaustive search like kNN, we make use of **approximate nearest neighbor search (ANN)** (Arya et al. [1998]). We retrieve a "good approximation" of the closest neighbor rather than measuring the distances between each vector in the database. In certain instances, accuracy might be sacrificed for increased efficiency. ANN allows to significantly improve similarity search performance when working with large datasets.

The **inverted file (IVF)** (Harman et al. [1992]) with k-means clustering is a cluster-based method solution to ANN problem. The inverted filed index links every word in the vocabulary to every instance it appears in the database, and k-means clustering is used to segment the dataset into partitions. This allows for faster and more efficient search times, as only a subset of the partitions need to be searched instead of the entire dataset. Additionally, using k-means clustering to create the partitions ensures that the partitions are balanced and evenly distributed, which can improve the accuracy of the search results.

In addition to cluster-based method, the graph-based method **HNSW** is also a powerful tool for vector similarity search. **Navigable Small World (NSW)** (Malkov et al. [2014]) is a proximity graph with both short-range and long-range connections. **Hierarchical Navigable Small World (HNSW)** (Malkov and Yashunin [2020]) enhances NSW by stacking multiple NSWs into layers to shorten the search paths and applying greedy search from a specific entry point on the graph. Unlike most proximity graph strategies, HNSW networks are entirely graph-based, which improves performance and enables logarithmic complexity scaling.

3 Pipeline

As shown in Fig. 1, the entire pipeline can be broken down into offline and online sections. The main goal of the offline section is the preparation of data for similarity search. We created image and text embeddings for the offline section initially using the CLIP model. The system then makes use of the embeddings and indexes them using similarity search algorithms.

The main responsibility of online section is responding to queries. Every time a query is received, the system computes the embedding also using the CLIP model that represents the query. Similarity search algorithms then use this embedding to search within the created index and retrieve the corresponding ones. After that, it will pull the image with these indexes from the database as the output.

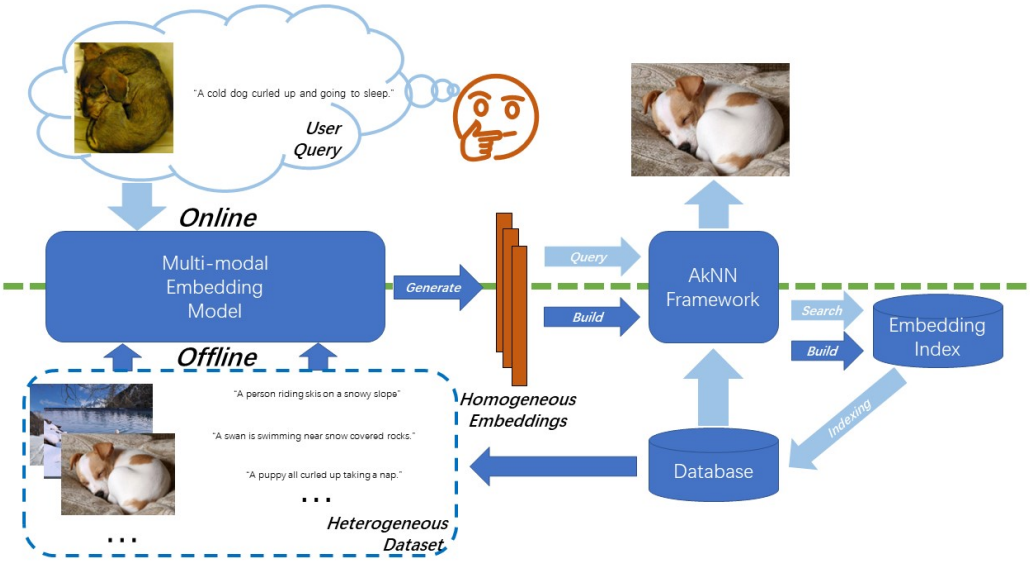

Figure 1: Embedding-based Graph Retrieval Pipeline

The success of the image-searching methods relies on their carefully designed data structure and searching algorithms, and we take full advantage of them in our pipeline. In order to efficiently search the most similar images for a given query, we use two libaries to index them: **Faiss (Facebook AI Similarity Search)** (Johnson et al. [2019]) and **HNSWLib** [1]. Faiss is a library for efficient similarity search applying ANN algorithm which has very efficient implementations of a few basic components like Inverted File Index and HNSW. HNSWLib is a lightweight version of Faiss that only supports the HNSW algorithm.

The underlying embedding data structure inside the similarity search algorithms plays an important role in both the online and offline sections. As shown in Fig.1, in the offline procedure, the similarity search algorithms take advantage of the result of feature engineering by building up the embedding data structure with the image and texts. The generated data structure will be maintained in the backend of our pipeline, waiting for queries. In the online procedure, the similarity search algorithms will do a similarity search within the built data structure to boost the search process. The query is first transformed into a vector embedding using CLIP. Then, from the index, we select the vectors that are most similar to the query vector.

4 Numerical Experiment

To evaluate the effectiveness and efficiency of our proposed pipeline, in this section, we first introduce the dataset, ground truth, and metrics that we used in experiments, then we will present our experiment design and results of the effectiveness and efficiency of the pipeline in out experiments.

4.1 Dataset Preprocessing

To evaluate our whole pipeline, we adapt MS COCO (Lin et al. [2014]) dataset considering both its content and scope. MS COCO dataset is a large-scale image dataset with 5 captions per image aiming at object detection, segmentation, and captioning. The captions in this dataset provide us with great correspondence between texts and images, which is a perfect example of heterogeneous datasets.

In experiments, we use the 2014 version of the MS COCO dataset, with a training set of size 82783 and a validation set of size 40504. For each example, we pass its image and corresponding 5 captions to CLIP model to get 512-dimension embeddings in the homogeneous embedding space and concatenate them into a 3072-dimension embedding.

[1] https://github.com/nmslib/hnswlib

To apply Faiss and HNSWLib, We did experiments with both L2 distances and Cosine Similarities to build the embedding index base on the embeddings of the training set and do the similarity search for 5000 examples randomly sampled from the validation dataset.

However, one drawback of MS COCO dataset is it doesn't provide ground truth for relevant items. We proposed a threshold-based solution that only the item in the training dataset that have a distance/similarity that is smaller/greater than a given threshold is considered as a relevant item for queries from the validation dataset. The threshold for Cosine Similarities is 0.75 and for L2 distances is 16, which guarantees that the number of ground-truth relevant items is approximately 300. We also limit the number of relevant items from 1 to 1000, which are sorted by the value of distance/similarity.

4.2 Metrics

We evaluate our pipeline with two aspects. For efficiency, we compare the searching time with indices built by different similarity search algorithms.

For effectiveness, we mainly focus on measuring Recall@k and NDCG@k. For each query, considering searching top k similar items in index, we define Recall@k as:

$$\text{Recall@}k = \frac{\#\text{Relevant in search result}}{\#\text{Ground Truth Relevant Item}}$$

Before defining NDCG@k, we first define the ranking score of a search result. For a query with a list of relevant items L, which is sorted by relevance in descending order and of size m, for an item e_j, the ranking score is calculated as:

$$G_{e_j} = \begin{cases} \frac{m-i}{m}, & \text{if } e_j \in L \text{ and is at position } i. \\ 0, & \text{otherwise.} \end{cases}$$

The NDCG@k is defined as following:

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{DCG}^{\text{ideal}}\text{@}k}$$

Where DCG@k and DCG$^{\text{ideal}}$@k is defined as:

$$\text{DCG@}k = \sum_{i=0}^{k} \frac{G_{e_i}}{\log_2(i+1)}, \quad \text{DCG}^{\text{ideal}}\text{@}k = \sum_{i=0}^{m} \frac{m-i}{m\log_2(i+1)}$$

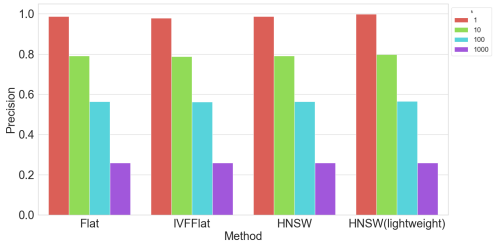

Figure 2: Precision with cosine metric

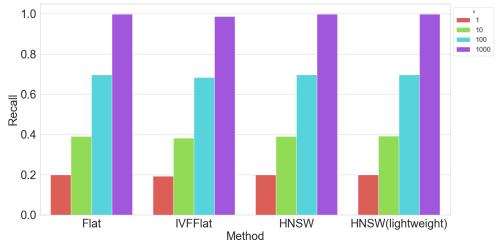

Figure 3: Recall with cosine metric

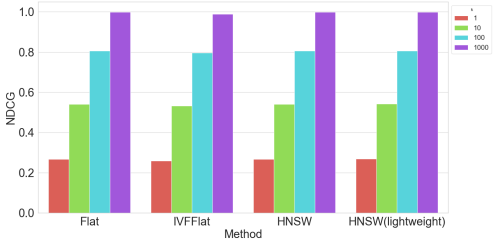

Figure 4: NDCG with cosine metric

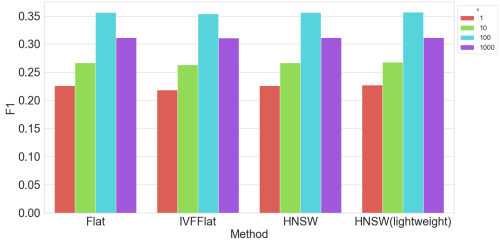

Figure 5: F1 with cosine metric

in which e_i is the i-th item in searching result.

We also consider Precision@k and F1@k in our experiments where:

$$\text{Precision@}k = \frac{\#\text{Relevant in search result}}{k},\ \text{F1@}k = \frac{2 \times \text{Precision@}k \times \text{Recall@}k}{\text{Precision@}k + \text{Recall@}k}.$$

For overall performance, we calculate the arithmetic mean of all the above metrics over every query.

4.3 Experiment Result

In this section, we display the result of our experiments as well as their analysis. In addition to the metrics mentioned above, we also present the searching time of each method, which helps us learn more about the trade-off between time and accuracy. Moreover, we conduct an in-depth investigation on the comparison between multi-modal and single-modal data inputs, which highlights this part.

4.3.1 Effectiveness

First, we compare the capability of four algorithms, Flat, IVFFlat, HNSW, and HNSW(lightweight), by calculating the evaluation metrics for $k \in \{1, 10, 100, 1000\}$ nearest neighbors. Flat is basically the simplest kNN method that conducts exact searches by comparing the query embedding and embeddings in the index one by one. HNSW is provided by Faiss while HNSW(lightweight), a lightweight version of HNSW, is provided by HNSWLib. Figure 2-5 present the result of Precision, Recall, NDCG and F1 scores with cosine similarities when k varies. The results under L2 distances are similar and thus placed in Appendix.

Flat can be regarded as a benchmark since it is bound to find all the relevant items. The performances of all the other algorithms are close to that of Flat, which indicates they all work. However, we can find IVFFlat is a little inferior to the other three algorithms since it only explores the clusters of interest which decreases its accuracy. However, HNSW, as an approximate method, performs as well

Figure 6: Query and its top 10 nearest neighbors

as Flat. In fact, the results of HNSW and Flat are completely the same while HNSW(lightweight) performs even slightly better than them. The reason is that we use the package sklearn (Pedregosa et al. [2011]) to calculate the ground truth labels which results in the mismatch between Faiss and HNSWLib.

Figure 6 presents a complete realization of our pipelines by HNSW(lightweight) that is to input a query and obtain similar returns. The actual input for each query is one picture and five captions. For simplicity, we only present the results in pictures. The first column is 10 different pictures. We can see that the returned 10 relevant items are semantically similar to the query. For example, the pictures on the fifth row all contain street signs.

4.3.2 Efficiency

Figure 7 demonstrates the searching time for a batch of queries by each algorithm. The time is an average result of cosine case and L2 case. First of all, the searching time has almost nothing to do with the size of k when the searching algorithm is determined. Next, HNSW requires twice the time as the other three methods to navigate hierarchy graphs. IVFFlat also needs more time to determine which cluster is of interest. Last but not least, HNSW(lightweight) is surprisingly quick since it is completely dependent on C++ code.

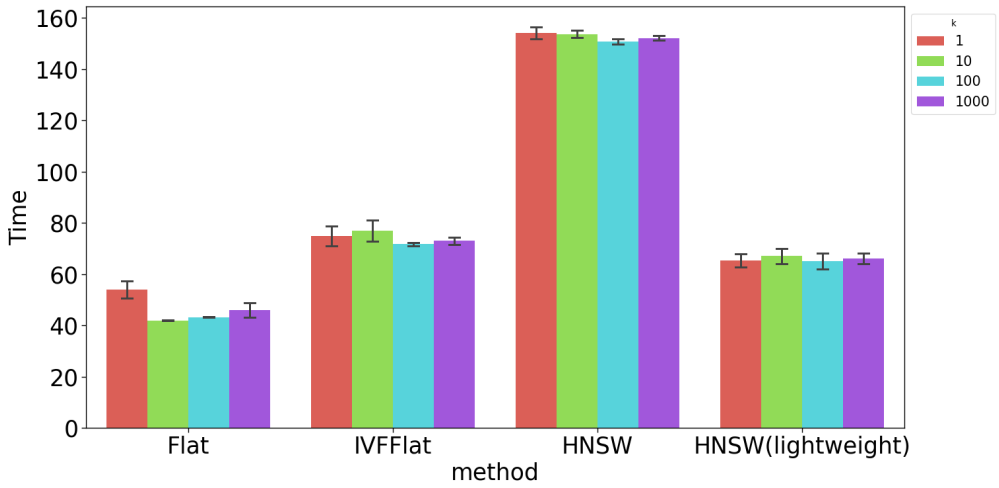

Figure 7: Average searching time

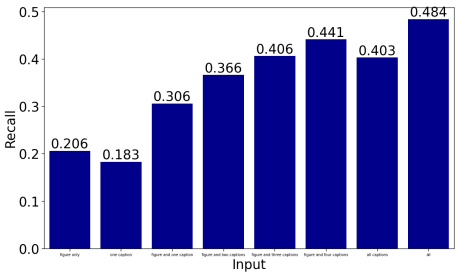

Figure 8: Recall for different inputs

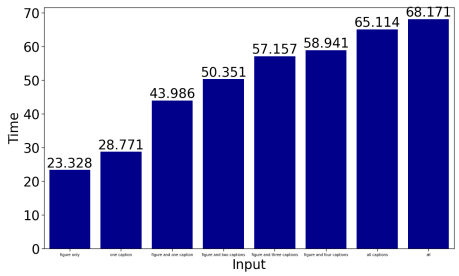

Figure 9: Searching time for different inputs

4.3.3 Multi-modal v.s. Single-modal Comparison

Based on the analysis of effectiveness and efficiency, we can see that Flat is the most suitable for small datasets while HNSW(lightweight) as an approximate method also performs fairly well. In order to better show the power of CLIP, we conduct experiments on inputs of different types, such as figure-only, text-only, and a combination of text and figure.

CLIP treats pictures and captions equally which means in the embedding, a picture takes up 512 columns and each caption also takes up 512 columns. Figure 8 and 9 represent the recall score and searching time needed respectively by HNSW(lightweight) when $k = 100$ with L2 distance. The results on Precision, F1, and NDCG are displayed in the Appendix. As for Recall, for a single picture and one caption, picture input has higher accuracy, even though they are the same amount of input. Then it is easy to understand that one picture with four captions has higher accuracy than five captions, while again they have the same amount of input. However, we find the astonishing fact that the input of one picture with three captions (0.406) also has higher accuracy than five captions (0.403). The interpretation is the combination of picture and text helps the neural network to learn better and thus have higher accuracy. With the picture and text combination, it uses less space and time as well to achieve higher accuracy as shown in Figure 9 where the figure with four captions (around 59s) takes less time than 5 captions (around 65s) even though they weigh the same.

5 Conclusion and Future Work

As shown in Section 4, we have done investigations in depth on the comparison among different combinations of data inputs and analyzed their effectiveness and efficiency. Comparison between HNSW and Flat search show that on small to middle-size datasets, approximate methods perform as well as exact search methods in light of similarity metrics and searching time. Besides, CLIP produces more semantically meaningful homogeneous embeddings with multi-modal data inputs than single-modal ones.

On the other hand, the dataset our experiment used is MS COCO dataset which has only 82,783 training data points. This is the reason why we found IVFFlat as an approximate method in our experiment does not perform well while flat search, the simplest method, has the smallest time complexity and the highest accuracy. However, when it comes to larger datasets in industry, things may go different. Hence, effectiveness and efficiency trade-off needs further analysis.

In the future, we would like to apply our implementation on large-scale dataset to test the scalability of this pipeline and to see whether the performance fits our expectation. We did not encounter bottleneck in memory because of the small dataset. In order to avoid such issue with large dataset, we may use Product Quantization(PQ) to improve the efficiency of similarity search and save valuable memory space.

References

S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6): 891–923, 1998.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

D. Harman, E. Fox, R. Baeza-Yates, and W. Lee. Inverted files. *Information Retrieval: Data Structures and Algorithms*, 1992.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

J. Heaton. An empirical analysis of feature engineering for predictive modeling. In *SoutheastCon 2016*, pages 1–6. IEEE, 2016.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2014. URL `https://arxiv.org/abs/1405.0312`.

Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2013.10.006. URL `https://www.sciencedirect.com/science/article/pii/S0306437913001300`.

Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2020.

J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Ng. Multimodal deep learning. In *International Conference on Machine Learning*, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th international conference on Machine learning*, pages 1168–1175, 2008.

A Appendix: Additional Plots

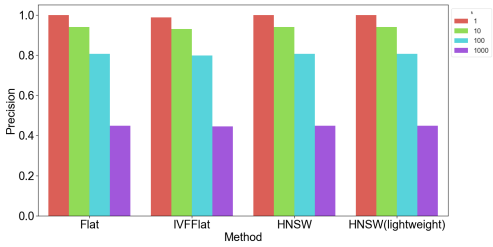

Figure 10: Precision with L2 metric

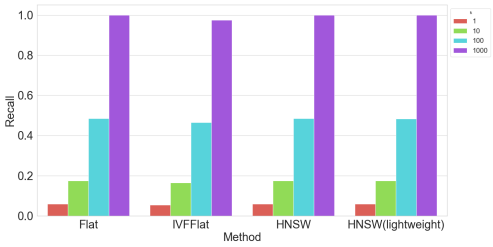

Figure 11: Recall with L2 metric

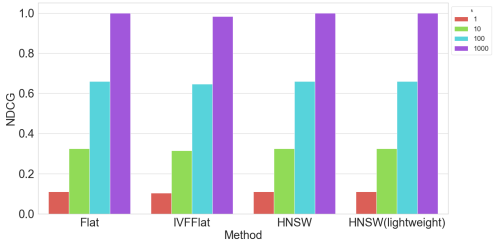

Figure 12: NDCG with L2 metric

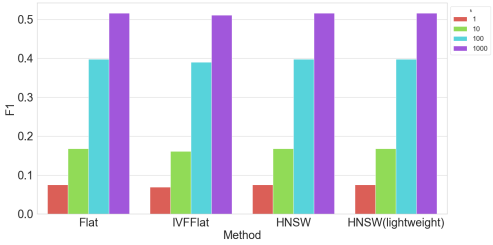

Figure 13: F1 with L2 metric

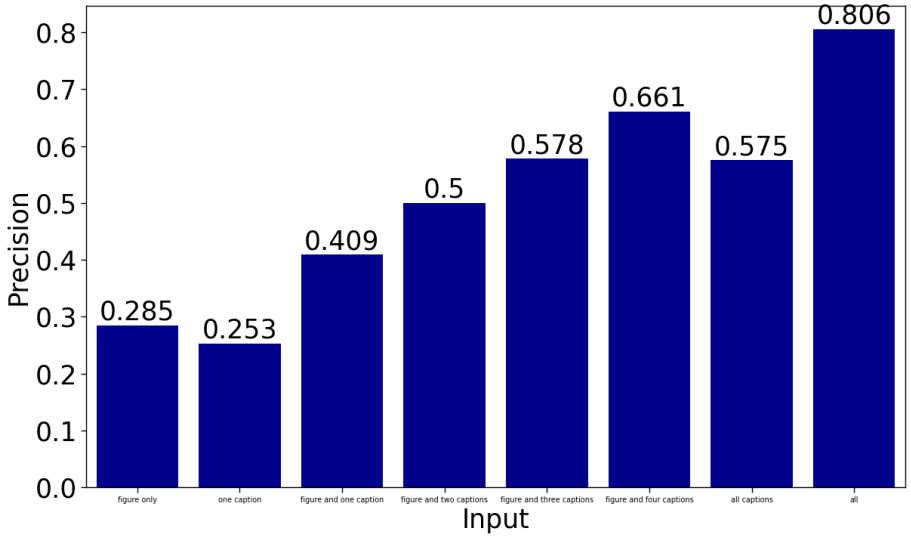

Figure 14: Precision for inputs of different sizes

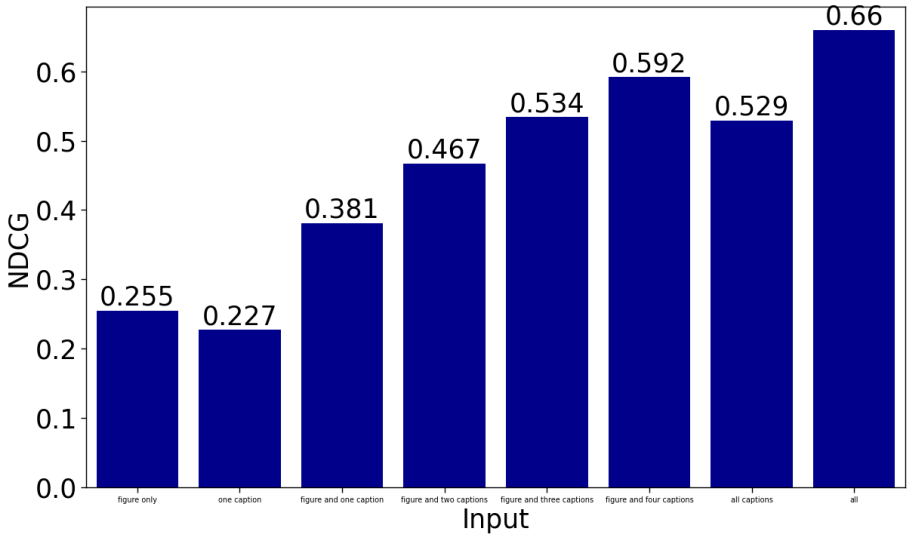

Figure 15: NDCG for inputs of different sizes

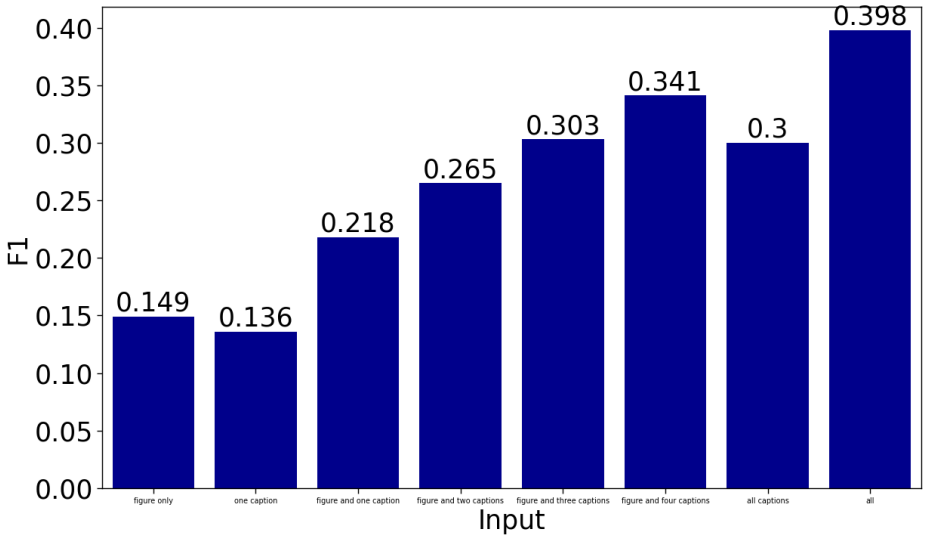

Figure 16: F1 for inputs of different sizes