FONDO TERRITORI LAVORO E CONOSCENZA CGIL, CISL, UIL

Deliverable D1.0

Clouds of vectors for density-based Information Retrieval

http://territoriaperti.univaq.it

Project Title	:	Territori Aperti

Deliverable Number	:	D1.0
Title of Deliverable	:	Clouds of vectors for density-based Information Retrieval
Nature of Deliverable	:	Report, Other
Dissemination level	:	Public
Licence	:	–
Version	:	1.0
Contractual Delivery Date	:	2023
Actual Delivery Date	:	2023
Contributing WP	:	WP3
Editor(s)	:	Giovanni Stilo (University of L'Aquila)
Author(s)	:	Andrea D'Angelo (University of L'Aquila), Giovanni Stilo (University of L'Aquila)
Reviewer(s)	:	Giovanni Stilo (University of L'Aquila)

Abstract

The report is organized as follows. Chapter 2 will list all the influences, models, and algorithms that made this work possible or that are somehow close to the proposed model. Section 3 will make an effort to explain all the intricacies of the model as a high-level overview. Implementation details will be instead documented in the subsequent section 4, which will also contain the environment in which the experiments were carried out. Results of such experiments will be shown and widely discussed in section 5. Lastly, section 6 will contain an in-depth discussion on the shown results and the knowledge gained by investigating, developing and analyzing the proposed model and similar ones, based on the gathered metrics during the testing phase.

Keyword List

NLP, Information Retrieval, Embeddings, Neural Networks

Table Of Contents

List Of Figures

1 Introduction

In the current time and age we use Information Retrieval systems multiple times a day, sometimes without even realizing we are operating one. One use case is, for instance, looking for something on the web, of course, but also searching for specific information on a web page or details of a certain celebrity, or a specific passage in a book, among many other examples.

The need for quick, readable at a glance information is ever-growing due to our massive use of portable and instantly accessible devices. Because of this behavioural tendency, in recent years, the attention of the field of Information Retrieval significantly shifted towards Passage Retrieval and technologies that involve a semantic representation of what the document is about. Such tools allow retrieving not only a relevant page, but also the contextualized information that the user is (likely) looking for, and presenting it in a nice and tidy way. Even Google has started adopting similar solutions in recent times, by presenting semantically contextualized information as tables or data as a first result, instead of web pages.

Keeping this context in mind, it should be no surprise that research in this field has drastically shifted towards the Transformers models which are able to embed contextualized semantics into each token. The popularity of this transformers neural network has been steadily on the rise during the past decade, and looking at ECIR 2022's (European Conference of Information Retrieval) proceedings [**?**], it is clear that their use is now widespread and has led to models claiming state of the art results for a variety of Information Retrieval tasks, with Google's BERT and Word2Vec leading the pack.

Their main purpose is to transform tokens into vectors that have their meaning, or semantic context, embedded inside. This allows for tokens that are part of the same domain to be transformed into tensors that are "close", according to any standard distance metric, between each other.

However, by computing the distances on the transformed tensors directly, information on the starting token is lost. This means, then, that standard techniques like a TF-IDF weighting schema are no longer possible since that would require a TF-IDF Matrix and memory on the tokens themselves. The most basic way to compute distances using tensors in this way would be by a Mean of Embeddings. This method involves reducing both the document and the query to a mean of the tensors they are composed of. Once the set of vectors has been aggregated to a single point, both for the document and the query, we can trivially compute the distance between the two points. Of course, this technique trivializes the matter and it is understandable to expect bad or unrelated results. It is not entirely meaningless, but it is too sensible to noise and will not correctly manage documents that contain more than one semantic topic.

The straightforward limits of this Mean of Embeddings baseline is the rationale that brought upon the model that will be detailed during this report work, called Cloud Retrieval, as it works with documents represented as sets of tensors, one for each token. Two algorithms will be presented for Retrieval on this model, named sentence-wide and document-wide. The former computes the distance metric for each query token on each separate sentence of the document, and aggregates the score afterwards. The latter, instead, computes the distance of the query tensor with respect to the Cloud of the document itself, without differentiating between sentences. Both algorithms have the same goal and are based on the similar intuition: computing the distance between each query term and a set of semantic-embedded vectors representing each document. The distance will be computed by a density measure.

The core idea is that if the query term is an "Outlier" with respect to the sets of vectors representing

the document, they are not semantically close and the query should therefore be ranked lower. If the query terms all have a low Outlier score, the query should, inversely, be among the top results.

The report is organized as follows. Chapter 2 will list all the influences, models, and algorithms that made this work possible or that are somehow close to the proposed model. Chapter 3 will make an effort to explain all the intricacies of the model as an high-level overview. Implementation details will be instead documented in the subsequent chapter 4, which will also contain the environment in which the experiments were carried out. Results of such experiments will be shown and widely discussed in chapter 5. Lastly, chapter 6 will contain an in-depth discussion on the shown results and the knowledge gained by investigating, developing and analyzing the proposed model and similar ones, based on the gathered metrics during the testing phase.

2 Background Knowledge and Related Works

As mentioned in section 1, Transformers and BERT in particular have dominated the landscape of recent progresses in Information Retrieval. Just by looking at the recent European Conference on Information Retrieval (ECIR 2022) proceedings [**?**], it is clear that much of the current research is actively pursuing semantic IR with the use of transformers. Many systems are able to achieve state-of-the-art results by employing and fine-tuning Google's BERT. A good summary of these projects is presented in [**?**], that details how BERT has been used with great success in question answering.

However, the field of IR presents a long history dating much earlier than the appearance of personal computers. In this section we will detail the most important models that played a crucial part for the development of the systems from which the proposed model has its roots.

2.1. Information Retrieval tasks

The comparison of our model with the baselines will be twofold: on one hand, we will compare our proposed model, Cloud Retrieval, to the basic score function called Mean of embeddings. On the other hand, we will assess its effectiveness as a re-ranking tool, by adjusting the score of documents that were already scored by BM25 and see if the metrics improve or worsen. An Information Retrieval tool can achieve many purposes: we will now present some of the most common tasks in literature, such that it will be clear in future chapters what we will be comparing our proposed model to and what is its intended use.

2.1.1. Ranked Retrieval

Ranked retrieval is by far the most common IR task, used daily by billions of users when looking for something on the Web through search engine services provided by Google, Bing, Youtube, etc. The first IR systems built used some variation of Boolean Retrieval, which meant that documents were either classified as Relevant or Not Relevant, with no score (and therefore no ranking) attached. With the inevitable evolution of these systems and the widespread usage of the Internet as a whole, Ranked Retrieval became by far the most important IR task, with Boolean retrieval systems relegated to very specific use cases.

The system has to compute a relevance score for each document in the collection and present the list of results with the document deemed "most relevant" on top, as the first result. This is intuitively what the user expects when interacting with a search engine. In IR literature the concept of relevance is not well defined, but it is widely assumed that a document is `relevant` if it satisfies the information need of the user that operated the search engine. Google does it very well and is generally considered reliable, even though the top results are often promoted or advertised for in some way. That is one of the many techniques used in Search Engine Optimization (SEO), which we will not be considering in the following. In particular, the score that Google assigns to each document (i.e., web page) is largely kept secret and allegedly changes periodically. We do know that part of the score is calculated by using techniques such as PageRank or some variation. The computational resources available to Google are second to none, and most of the scores are saved and re-computed every once in a while. Our system, instead, recomputes all the scores every time a query is entered into the system.

2.1.2. Passage Retrieval

Passage Retrieval is a subset of Information Retrieval that specializes in finding not documents, but specific passages or sentences inside the documents that might directly answer the user's query. It is not meant as a complete replacement of the classic ranked retrieval, as entire documents are still much more useful than specific passages in a wide spectrum of situations and purposes. However, many search engines have started to subtly provide Passage Retrieval functionalities when users enter specific queries, such as an actor or actress' age, the release date of a particular movie, or similar questions. For queries like those, displaying the specific information without returning the entire document can be very handy.

Much of the current effort in research is being channeled towards Passage Retrieval. Even by looking at ECIR 2022 conference proceedings [**?**], we can see that many of the proposed models and algorithms are part of this domain. Some of these works also include semantic-embedded vectors computed by Transformer Neural Networks such as BERT or Word2Vec.

2.1.3. Reranking

The reranking task is fairly simple and consists of re-arranging results that were already ranked by an external IR system, with the goal of improving the relevance of the presented results. This is a sort of search pre-processing: the top results from a certain system are taken, their store modified in some way, and then they are sorted again based on the new score. This will be one of the tasks that the new model will be tested on. Specifically, the ranked results of BM25 will be taken and re-ranked. More details can be found in section 5.

2.2. Vector space model

In classical Information Retrieval, the vector space model is how documents are usually represented. Simply put, each document will be characterized by a single vector of n dimensions, where n is the size of the vocabulary in the collection.

Therefore, for document i, we will have a vector such as:

$d_i = (w_{1,i}, w_{2,i} \dots w_{n,i})$

where each $w_{j,i}$ corresponds to a separate term. In Boolean retrieval, we would have 1 if the term appears in the document and 0 if it does not. In more modern and sophisticated models, we would instead have the TF-IDF score of that term in the document.

2.2.1. TF-IDF function

The TF-IDF weighting schema is defined as:

$$TF - IDF(t,d) = TF(t,d) * IDF(t,D) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} * \log \frac{N}{|\{d \in D : t \in D\}|}$$

which is basically the product between the relative Term Frequency TF of that term t in document d and the inverse document frequency of the term t in the collection D.

The idea behind the TF-IDF function is to assign an high score to a term that appears frequently in the document (indicating that the document focuses on that concept) but not too many times, because in that case it might be a stop-word or a word that is not really meaningful. Its use is now widespread among classical Information Retrieval systems and has been the standard for a long time.

There is no strict definition for stop-words, but it is generally a word that is extremely common and holds no particular meaning. Great examples are words such as "in", "if", "for", "a", "what", etc...

2.2.2. Distance metrics

In order to retrieve ranked documents, standard IR systems encode the query in the same way as the document. The query can thus be defined as

$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$

and this allows to consider multiple options for distance metrics. The most natural one would be to just compute the euclidean distance between each document and the query. However, the most popular solution by far is the one of **cosine similarity**. With cosine similarity, we compute the cosine of the angle between the vector representing the query and the ones representing the documents:

$$\cos(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

Recall that no element of the document vectors or the query vector is negative, since they were defined using the TF-IDF weighting schema (or a Boolean schema in earlier models). Because of this, a cosine similarity of 0 implies that the document vector and the query vector are orthogonal, so they have no match whatsoever. The higher the cosine similarity, the higher the document is ranked among the results.

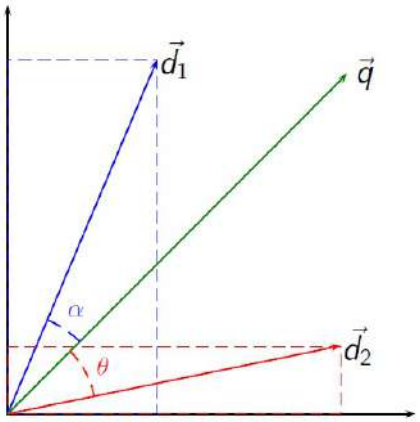

Figure 2.1: Example of cosine similarity between query and two documents.

A visual representation of the cosine similarity metric is shown in figure 2.1. Naturally, this figure either implies that the figure has been simplified or that there are only two terms in the collection, since they were plotted in two dimensions. It is not easy to visualize results of this kind when dealing with a realistic collection.

2.2.3. Okapi BM25

BM25 is a commonly used ranking function (BM stands for Best Matching) for Information Retrieval systems such as search engines. The full name, Okapi BM25, actually includes the name of the first method that made use of it, the search engine Okapi. Recent systems still include BM25, among other technologies such as BERT [**?**].

BM25 uses the `Bag of words model` to retrieve and rank documents based on the query terms that appear in each document. It basically uses an edited version of the TF-IDF score to assign a custom score to each document.

BM25 will be the baseline against which we will compare the re-ranking properties of the proposed model. More details will be presented in section 5. We decided to use BM25 because it provided efficiency, good results, and it is widely used in the Information Retrieval literature.

2.3. Transformer Neural Networks

Transformer Neural Networks were first introduced in the Attention is All You Need paper [**?**], that argued how dropping the recurrent nature of the former state-of-the-art models to focus exclusively on Attention Functions produced better results while speeding up training considerably, due to the new-found opportunity for parallelization. In particular, an Attention Function as defined in the aforementioned paper maps a Query and a Set of Key,Value pairs to an output, where all of the involved elements are Vectors.

The computed output is a weighted sum of the values, where the assigned weight to each value is the result of a Compatibility function between the Query and the Key.

Only two years later, in 2019, Google published the famous Google BERT paper [**?**], presenting their new Language model capable of outstanding performances. By allowing the end user to fine-tune it, BERT is extremely customizable and flexible and has thus been used by several systems to obtain state-of-the-art results in many subfields of Passage Retrieval. BERT is still a transformer, so the way it works is similar to what was previously mentioned: given a collection of documents, BERT transforms each token or sentence into semantically embedded vectors. What this means is that the same token will be transformed into drastically different vectors if it appears in different contexts. This is crucial to understand the model that we are about to propose.

2.3.1. Mean of embeddings

The most basic way to use the embeddings made available by a transformer Neural Network would be to compute the distance between the mean of the tensors that make up each document with the mean of tensors that represent the query.

In other words, the score for each document is computed with:

$$Score(document) = distance(\frac{\sum_{i \in document} tensor_i}{|document|}, \frac{\sum_{j \in query} tensor_j}{|query|})$$

This will be the baseline that we will compare our model to. Of course the mean of embeddings is the most intuitive approach imaginable, so we expect our model to outperform it in every metric to have significance.

2.4. Outlier metrics

The aim of our model is not only to consider the implementation of Transformer Neural Networks to rank documents. The true novel approach is to use a density-based metric to find which documents are the most relevant to the query. Details will be discussed in section 3, but references for such metric were many outlier metrics, including Local Outlier Factor LOF [**?**].

Our goal was to score a specific vector given its closeness to clusters of other vectors, or in other words, based on how much the density around it is similar to the density of vectors around the others. This score would give us a numerical answer on how much of an outlier that vector is in respect to the others. We found that the Local Outlier Factor LOF, while not being exactly what we needed, was a good starting point.

Another Density-Based outlier metric is DBSCAN [**?**]. It is by far the most commonly used in literature, and simply put, it connects regions of points with high density. In particular, DBSCAN defines a cluster as:

[Cluster]

Let D be a database of points. A **cluster** C with respect to ϵ and MinPts is a non-empty subset of D satisfying the following conditions:

1. $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt. ϵ and MinPts, then $q \in C$

2. $\forall p, q \in C$: p is density-connected to q wrt. ϵ and MinPts.

This definition makes use of some terms, like density-reachable and density-connected, that needs to be defined as well.

[ϵ-neighborhood]

The ϵ-neighborhood of a point p, denoted by $N_\epsilon(P)$, is defined as: $N_\epsilon(P) = q \in D|dist(p,q) \leq \epsilon$

[Direct density-reachability]

A point p is directly density-reachable from a point q wrt. ϵ, MinPts if:

$p \in N_\epsilon(q)$ and $|N_\epsilon| \geq$ MinPts

where MinPts is the minimum amount of points that must be inside the ϵ -neighborhood of a point inside a cluster.

[Density - reachability]

A point p is density-reachable from a point q wrt. ϵ and MinPts if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$ and $p_n = p$ such that p_{i+1} is density reachable from p_i.

These definitions are all fairly simple. In order to fully understand definition 2.4, we still only need to define Density connectivity:

[Density - connectivity]

A point p is density-connected to a point q wrt. ϵ and MinPts if there is a point o such that both p and q are density-reachable from o wrt. to ϵ and MinPts.

2.5. Evaluating an Information Retrieval model

At its core, every IR system's goal is the same: retrieving information that the user finds relevant according to their query, among a collection of documents. There are a plethora of metrics who aim to evaluate such a system. The most basic ones are **Precision and Recall**.

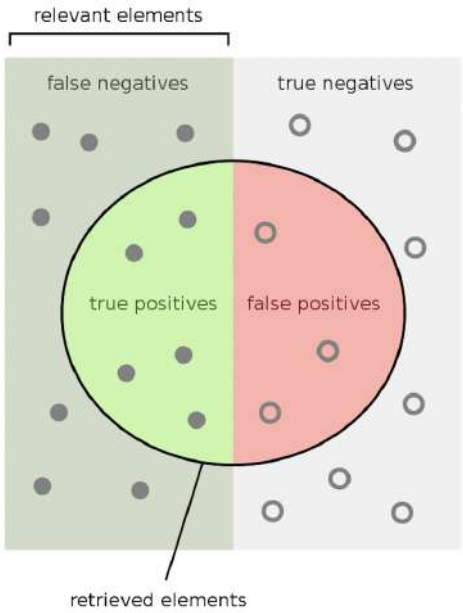

Figure 2.2: Set of retrieved vs. relevant documents

Defining True Positives (TP), False negatives (FN), True Negatives (TN) and False Positives (FP) as in figure 2.2, we can then identify precision and recall as:

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

Another very important metric is the Mean Average Precision mAP. mAP is defined, of course, as the mean of the average precision of all classes.

$$mAP = \frac{1}{n} * \sum_{k=1}^{n} AP_k$$

There are several tools forged with the purpose of evaluating IR systems, for instance **trec_eval**. Trec eval asks for the collection's qrels and the results, given in a specific format. A collection's qrels is a dataset composed of a sample of specific queries embedded with the hand-picked relevant documents for each. They provide a stable ground-truth for relevant documents relating to a specific query and allows the system to be objectively evaluated. We will present more details in section 4.

Trec_eval computes and presents a lot of metrics that are a good indication of the Information Retrieval system's performances with respect to the given queries. In particular, the measures that will be provided are, among others:

run_id	Name of the run.
num_q	Total number of evaluated queries.
num_ret	Total number of retrieved documents.
num_rel	Total number of relevant documents, according to qrels files
map	Mean average precision.
gm_map	Geometric mean of average precision
Rprec	Precision of the first R documents, where R are the number of relevant documents.
bpref	Binary preference.
recip_rank	Reciprical Rank.
iprec_at_recall_0.00	Interpolated Recall - Precision Averages at 0.00 recall
iprec_at_recall_0.10	Interpolated Recall - Precision Averages at 0.10 recall.
P_5	Precision of the first 5 documents.
P_10	Precision of the first 10 documents.
P_30	Precision of the first 30 documents.
P_100	Precision of the first 100 documents.

3 Through the Clouds of vectors and beyond (Density-based ranking models)

3.1. Documents as clouds of vectors

While Transformers are fairly common in literature, representing documents as a set of vectors is far less popular. To the best of our knowledge, the only works that moved in this direction are an earlier version of this project [**?**] and a paper involving the Word2Vec embedding [**?**]. In particular, Word2Vec is a simple two-layer Neural Network which aims, like BERT, to embed semantics into a vector representation of the tokens. The research then uses a score calculated by a probabilistic function, and distances itself considerably from our work, where we propose a density function to be used as distance metrics. We want to emphasize that, despite the entire Transformer technologies being fairly recent, the true novel approach is a byproduct of this density metric.

Representing a document as a set (from here on, ”cloud”) of vectors is drastically different to the classical IR approach of representing a document with a vector of n dimensions, where n is the number of terms in the collection. It allows us to avoid using sparse vectors and focus on few dimensions that are significant. Of course this kind of model also helps us with the visualization of concepts and distance metrics. While a single vector with thousands of dimensions might be hard or even meaningless to visualize, the same is not true with a Cloud of vectors representing tokens, which might give us a good idea on how the document is structured and whether or not it contains multiple topics, semantically distant from each other.

Referring to figures 3.1 and 3.2, these images represent a document that was transformed by BERT and then visualized by applying PCA on its vectors. It is easy to see that this document contains three clusters of semantically related terms, and we can thus infer that it focuses on three main topics. Naturally, this is just a basic example, but it is helpful to understand the idea behind the model at a glance.

Given a collection of documents, we want to be able to perform Information Retrieval tasks such as re-ranking and Document retrieval by transforming documents into a Cloud of vectors with embedded semantics and then query this collection by retrieving the documents that are nearest to the query according to a certain density metric.

The rationale is that a document might be identifiable by a set of clusters denoting the semantic topics it is relevant about. Therefore, if the cloud of vectors is not an Outlier with respect to the documents, it means that the document contains those same topics as the queries and should therefore be ranked highly within the Ranked Retrieval. Naturally, this comes with the assumption that semantically related terms will be transformed into tensors that are close to each other.

This is, in a way, an opposite approach to Retrieval with respect to the tf-idf metrics. Documents are likely to be deemed Relevant to a query even if they share no actual terms with said query, if they are enough semantically close to its topics.

This is the summary of our model. Any tools, be it with BERT or with any other transformer, or metrics like LoF or any other outlier metric, is just a matter of specific implementation and should be tested accordingly.

For simplicity, the model will be partitioned into two main sections: the Cloud of Vectors model of

representation and the Query algorithm. We will see the former in-depth first, while also keeping a high level of abstraction. Please refer to section 4 for implementation details, including which technologies and metrics were chosen to test the model.

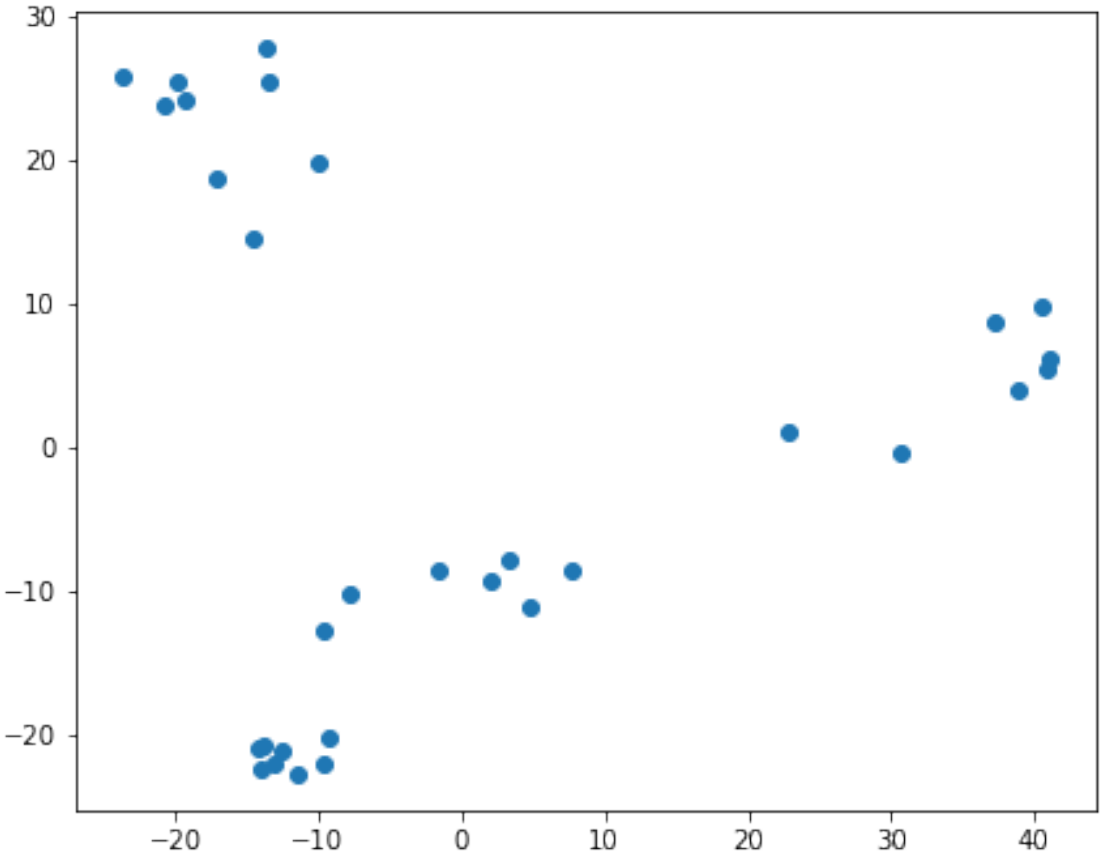

Figure 3.1: Example of a cloud of vectors representing a document.

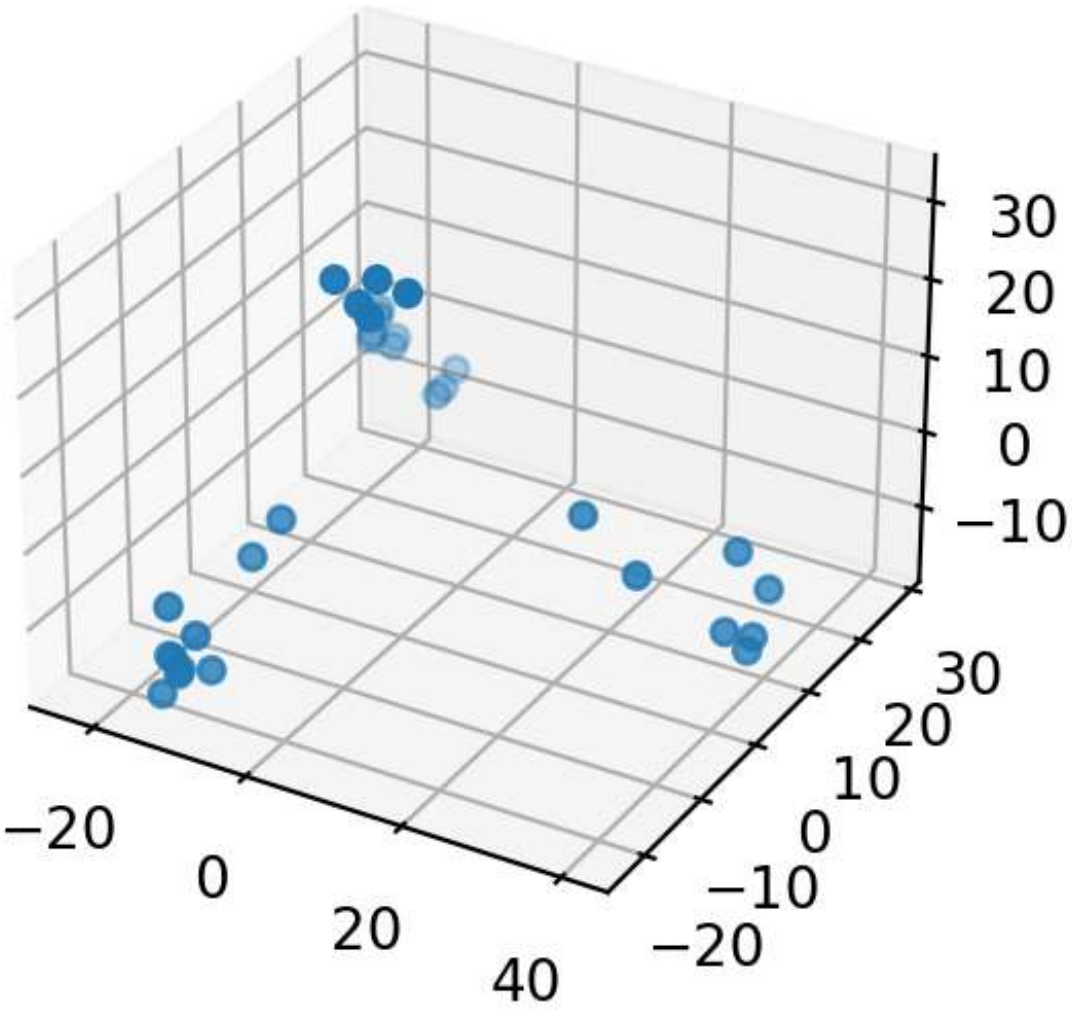

Figure 3.2: Example of a cloud of vectors representing a document, in three dimensions.

3.2. Cloud of vectors model

The first section of our model is dedicated to transforming the collection of documents into a collection of set of vectors. In particular, each document will be transformed into a set of vectors with embedded semantics: this can be done with the Transformer Neural networks presented in section 2, including (but not limited to) Word2vec and Google's BERT.

The vectors' sizes will depend on the specific neural network architecture and, in particular, the size of the output layer. What matters the most is that the number of dimensions will be fixed, and will certainly not be as huge as what we would have in a Classical information retrieval model, thus avoiding the so-called Curse of Dimensionality. This term was first used by Richard E. Bellman and indicates the problem of having vastly sparse models when the dimensionality increases. This prevents data organization strategies from being effective, including clustering and outlier detection, which will be of cardinal importance to our model.

This approach also presents one great advantage with respect to the classical IR vector space model. We can choose to save information on the sentences composing the documents so that we can then define exactly which vectors were taken from certain sentences, which is lost information when transforming the whole document into a single vector.

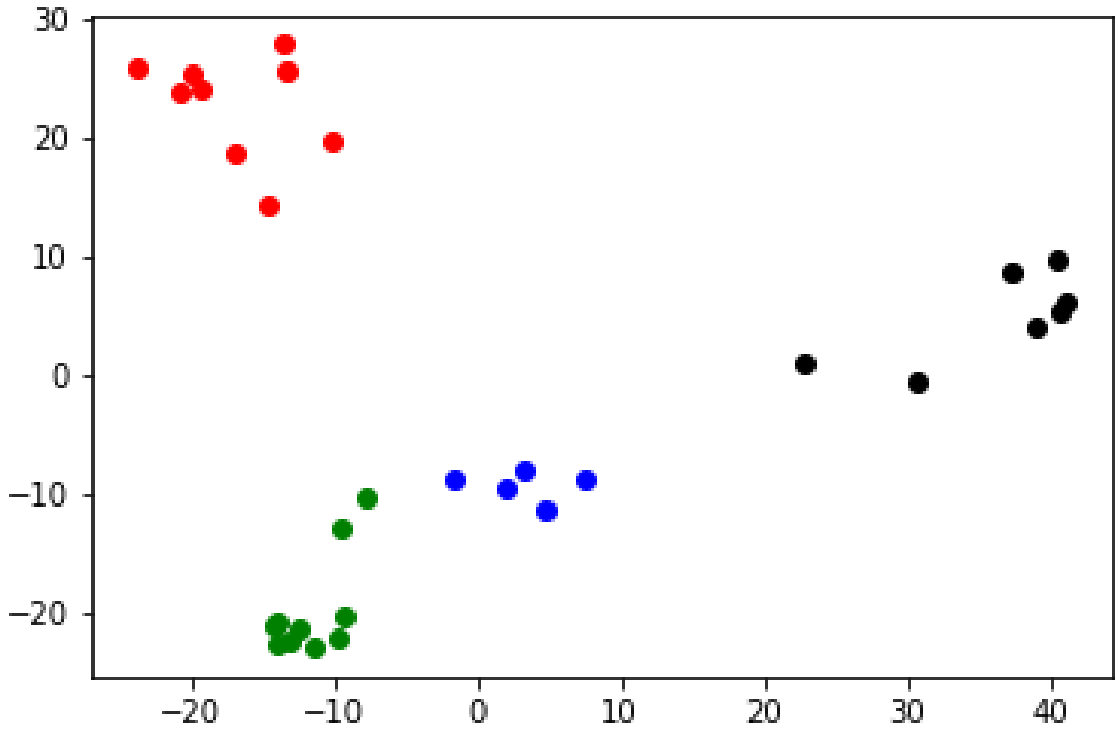

Figure 3.3: Cloud of vectors colored differently for each origin sentence

Let's take figure 3.3 for reference. A visualization of this kind can be extremely helpful when studying Passage Retrieval instead of Information Retrieval.

Passage retrieval is a subset of IR focused on retrieving a certain sentence, passage or meaning from a collection rather than a ranked set of documents.

Of course, this is also possible in classical IR if we diminish the granularity of a "document" to a single sentence, obtaining one vector for each sentence. That, however, will not only lead to a completely

different model, but also to even more sparsity in the resulting vectors. The cloud of vectors instead does not change, we only have to record the origin sentence of the vectors.

It is easy to see how being able to distinguish which sentence the vectors originated from, and consequently being able to draw a distance metric for each sentence individually, can be of great help for such a task. This is just an example of the high versatility of the proposed model.

3.3. Query algorithms

Once the documents have been transformed into clouds of vectors, we can finally call the query algorithm to find the most relevant documents for a specific query. It is important to note that, in the classic proposed algorithm, at this point no such thing as ”terms” or ”documents” remain: we will only work with clouds of multi-dimensional tensors. This is a novelty proposed by our method: from here on, only vectors and a selected density metric between them will be considered.

This proposal stems from the observation that semantically similar tokens will be clustered into nearby vectors, thus generating high density zones. In this context, let's consider a query composed of one single token, transformed into one single vector, and compare it to a document. If the query vector is an outlier with respect to the clusters of the document, then we can assume it is not relevant to those semantic clusters, and the latter should be ranked lower. In general, the higher the ”outlier score” of a query vector with respect to a document, the lower the document should be ranked.

If a document generates more than one cluster, intuitively, it means that it contains more than one topic. This is where our model shines with respect to the classical one: it will rank the document high if the query strictly relates to one of the topics (since the outlier score will be on the low side) but is not relevant to the others, a feat that the vector model struggles with.

If the query is composed by more than one term, which is often the case, the query algorithm will compute the outlier scores for each query term one at a time, and aggregate them via an aggregation function at the end to obtain the total score of the query for each document, and rank them accordingly.

Let us assume that the query is composed of n terms, so $q = q_0, q_1, \ldots, q_n$.

We will refer to the density metric between a document and a query term as Outlier(d, q_i).

First of all, we will present the basic query algorithm that computes the query term outlier score for each sentence of the document Outlier$(sentence, q_i)$, and then computes Outlier(d, q_i) as an aggregation of the former.

Algorithm 1: Query algorithm, sentence-wide

for *Document d ∈ Collection* **do**
  $q_i \in Q$ **for** *Sentence s ∈ d* **do**
  Compute Outlier(s, q_i)
  Compute Outlier(s_i, q) =AggFunc1(Outlier(s_i, q_i), . . . , $Outlier(s_i, q_n)$)
  Compute Outlier(d,Q) = AggFunc2(Outlier(s_1,q), ..., Outlier(s_k, q))
  | $s_1 \ldots s_k \in d$
  Rank Outlier(d,Q) ∀ d ∈ collection

In this algorithm we have two Aggregation functions that can be defined in various ways (instances will be provided in section 4). The first one aggregates results for all the sentences of a document into a score for that very document, while the second will aggregate the document score for all query terms, obtaining the final result of one single score for each document with respect to the given query.

If we instead choose to directly relate the query vectors to the cloud of vectors representing the document, without saving information about the single sentences, we will of course skip one of the aggregation functions, as shown in algorithm 2

It can be easily noted that the collection of documents that the algorithms use can be whatever. In

Algorithm 2: Query algorithm, document-wide

for *Document d ∈ Collection* **do**
$q_i \in Q$
Compute Outlier(d, q_i)
Compute Outlier(d,Q) = AggFunc2(Outlier(d,q_1), ..., Outlier(d,q_n))
Rank Outlier(d,Q) ∀ d ∈ collection

the case of classic IR, we would like to give a dataset of documents and let the query algorithm retrieve the most relevant ones with respect to a given query from the user. Summing up, the general algorithm needs an Outlier (or density) function to score the query term vectors, and potentially two aggregation functions for the sentence-wide version.

As for deploying the system as a `re-ranking tool`, the process is intuitive and easy: the collection to be given as input will be the set of the top N documents retrieved by the system of choice (in our case, BM25). The score computed by Cloud Retrieval will then be summed to the original score, and the documents sorted once again.

Algorithm 3: Reranking algorithm, sentence-wide

for *Document d ∈ Top N results* **do**
$q_i \in Q$ **for** *Sentence s ∈ d* **do**
Compute Outlier(s, q_i)
Compute Outlier(s_i, q) =AggFunc1(Outlier(s_i, q_i), ..., $Outlier(s_i, q_n)$)
Compute Outlier(d,Q) = AggFunc2(Outlier(s_1,q), ..., Outlier(s_k, q))
| $s_1 \dots s_k \in d$
Rank $(1-\alpha)$ ⋆ OriginalScore(d) + α ⋆ Outlier(d,Q) ∀ d ∈ collection

It should be noted that the Cloud Retrieval score, called Outlier(d,Q), will be weighted by a certain factor α. Therefore, the final score will be a linear combination of the original score and the new score, rather than a straight sum.

The choice for this variable α is not straight-forward. If it is set as too small, the new score will be ineffective and will change nothing of the original score, thus erasing our reranking efforts. On the other hand, if α is too big, the original score will be annihilated by the new one, thus turning it back to a straight Ranked retrieval task. Results with multiple choices of the parameter α will be presented in section 5.

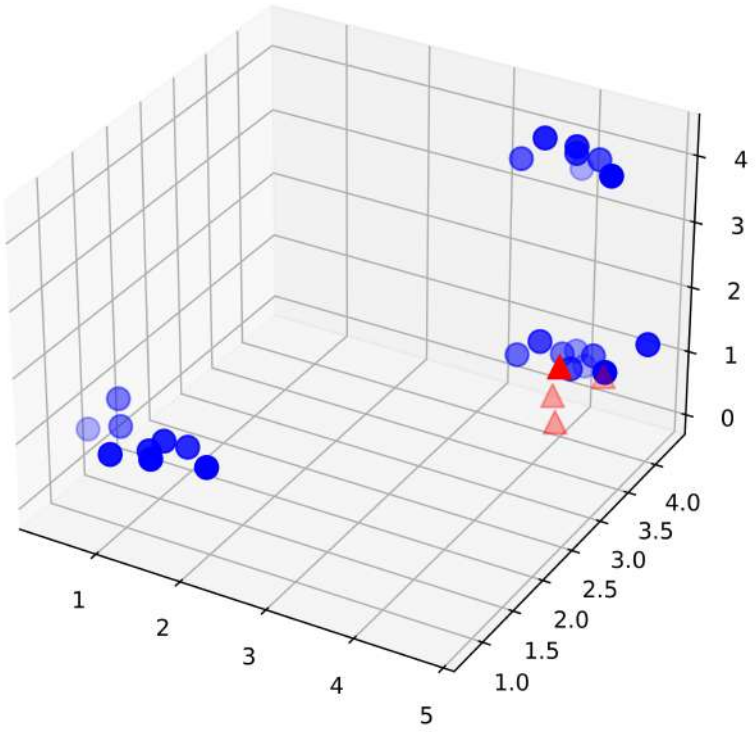

Figure 3.4: Example of transformed query with a transformed document.

Let's take figure 3.4 as an example. The document is plotted as blue dots, while the query is plotted as red triangles. Each red triangle will be treated individually and its Outlier metric computed with respect to the whole sentence, or document. The final score will be the output of an aggregation function that takes as input the score for each query term. The ideal result, in this situation, would be having this particular document as high ranked in the final results, because the queries are intertwined closely with one of the topics of the document. While such a situation would be penalized in a classical IR system since many of the other tokens are probably very different from the query, we expect Cloud Retrieval to reward the document in this case.

4 Implementation and Technical Details

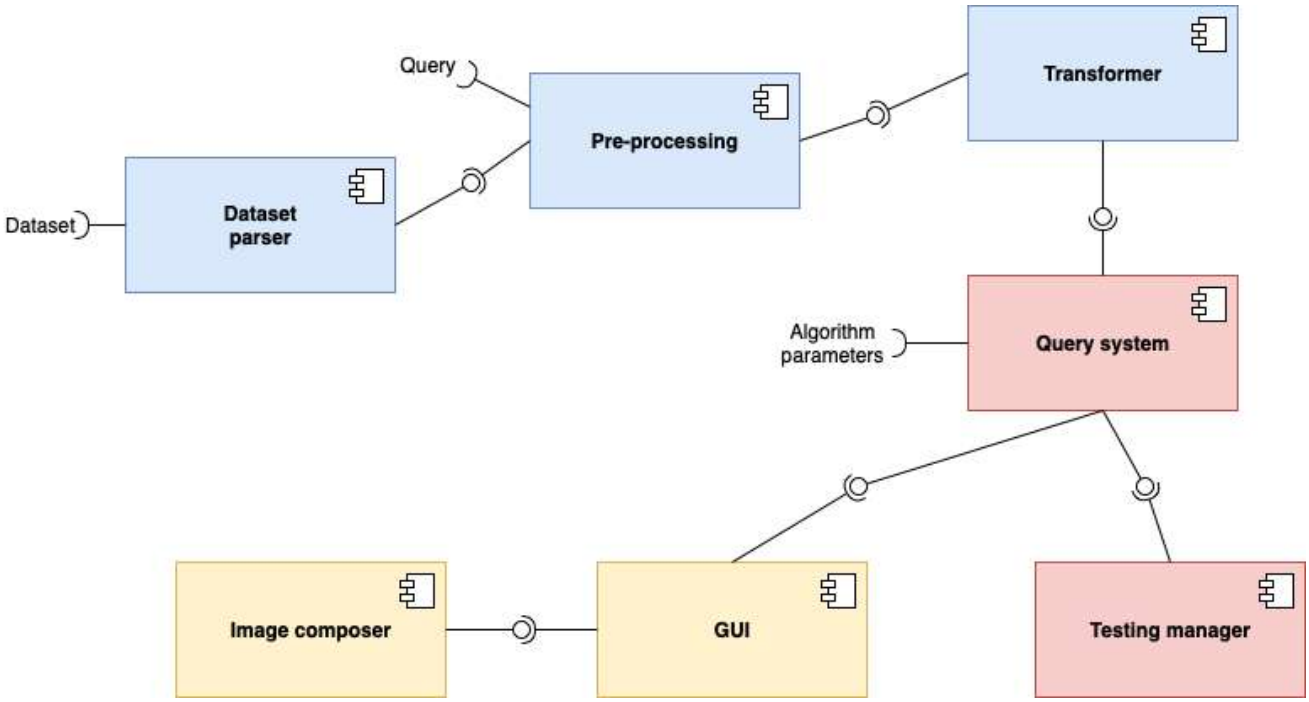

Figure 4.1: Component diagram of the implement system.

This section will document the modules of the implemented system and the design choices that were made in an effort to test out the reference model, described in section 3.

Figure 4.1 illustrates the components of the resulting system, color-coded based on their main purpose. Blue components process and transform the dataset from raw to a set of semantic-embedded vectors.

Red components manage the various processes whose aim is to query the system, with the requirement of being as efficient as possible. Lastly, yellow components manage an on-going project of Graphical user interface for the query system, built both to simplify early testing and as an end-user interface.

4.1. Dataset parser

The first module takes as input the given, raw dataset, and tries to format it correctly for the upcoming steps. In particular, the raw dataset will be formatted as a Pandas Dataframe with columns {doc_id, text} and a row for each document. This module is the only one that should be customized based on the dataset at hand in order to obtain the desired result. Once the dataset appears in the aforementioned format, the other modules will take care of all the subsequent steps without any modification needed.

4.2. Pre-processing

The dataset's corpus is then pre-processed with standard techniques. The following steps are executed:

- All the documents' text is transformed in lower case.
- Stop words are removed from the corpus. The list of stopwords is taken from the sklearn.feature_extraction library.
- The documents are split into sentences and saved into a different file, where info on the different sentences is maintained.
- Empty sentences or sentences with less than k tokens are removed.
- A tf-idf matrix of the whole collection is computed for later use.

while modern vectors space models do not make use of stop lists, it is very helpful for our model to part ways with them, since it considerably speeds up the transformation process and helps outlier detection removing unnecessary noise from the clouds of vectors.

4.3. Transformer

This module operates on the main part of the Cloud of vectors model presented in section 3. Each sentence is individually taken and used as input for the Transformer Neural Network, which will transform it into a set of vectors with embedded semantics. Alternatively, the whole document can be taken and given as input, without maintaining information on the single sentences.

This step will produce a series of files where the vectors, along with the reference document, are saved. Of course the files represent a complete partition of the starting collection. Please refer to figure 4.2.

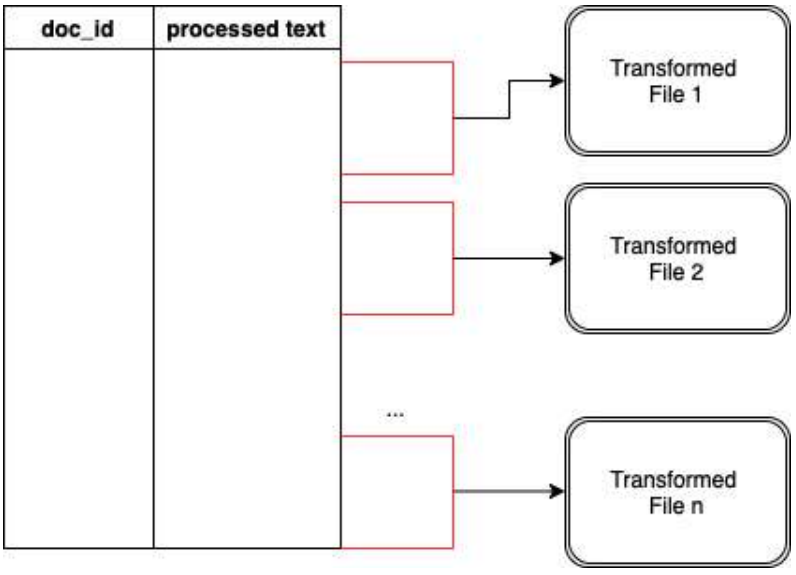

Figure 4.2: A pre-processed collection is transformed into a set of file containing clouds of vectors.

The number of documents contained inside each transformed file is easily interchangeable and should be decided based on the currently available computational resources.

For this particular implementation, we decided to make use of Google's BERT, as mentioned in section 2.

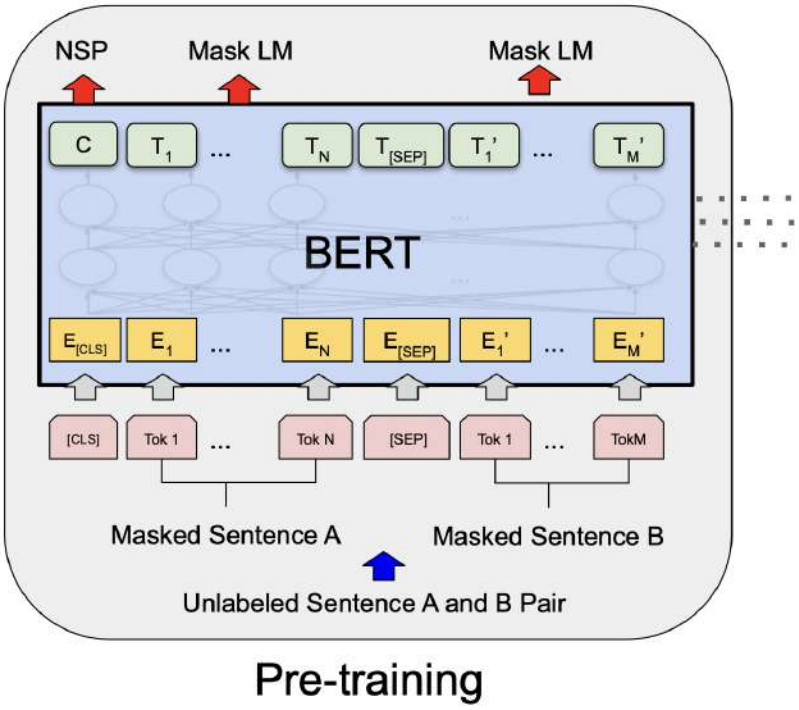

Figure 4.3: BERT architecture.

BERT has 12 hidden layers and 768 hidden units, and there are multiple ways to obtain vectors for each token **without any fine-tuning**, by only using the pre-trained model. For instance, the last n layers could be concatenated. In our solution, we opted to sum the last 4 layers.

Each token can be represented by a [12 x 768] vector, which is just the amount of hidden layer times the amount of hidden units as previously mentioned. We opted to sum the values of the last 4 layers of the Neural Network to obtain a 768-dimensional vector for each token.

This is an easy solution, but the question is still being investigated in literature as of now. Multiple answers could and will be investigated for this project. One other option could be to **fine-tune** the model, an alternative that we initially discarded for simplicity purposes.

4.4. Query system

During query time, each transformed file is managed by a different python script (all executed in a parallel way).

Therefore, each process ranks the clouds contained in each transformed file, which are a fraction of the whole collection. Once all the processes have finished, an aggregation process computes all the results and composes the final ranking.

It is important to note that, in order to exploit the available resources as much as possible, these are not python processes managed by a single python script (since multi-threading in python is a bit lackluster and ambiguous), but are literal different python scripts that communicate via the file system. Once all of them are done, the result folder will contain n amount of result files. Therefore the new aggregation process can start and compute the final ranking, by removing the folder and creating a new, final file.

We will now describe in greater detail how the individual ranks for each file are computed. Please refer to Algorithm 1. As mentioned, in order to implement the algorithm we need to define an Outlier metric and two different aggregation functions. The possibilities are endless, but the design choices

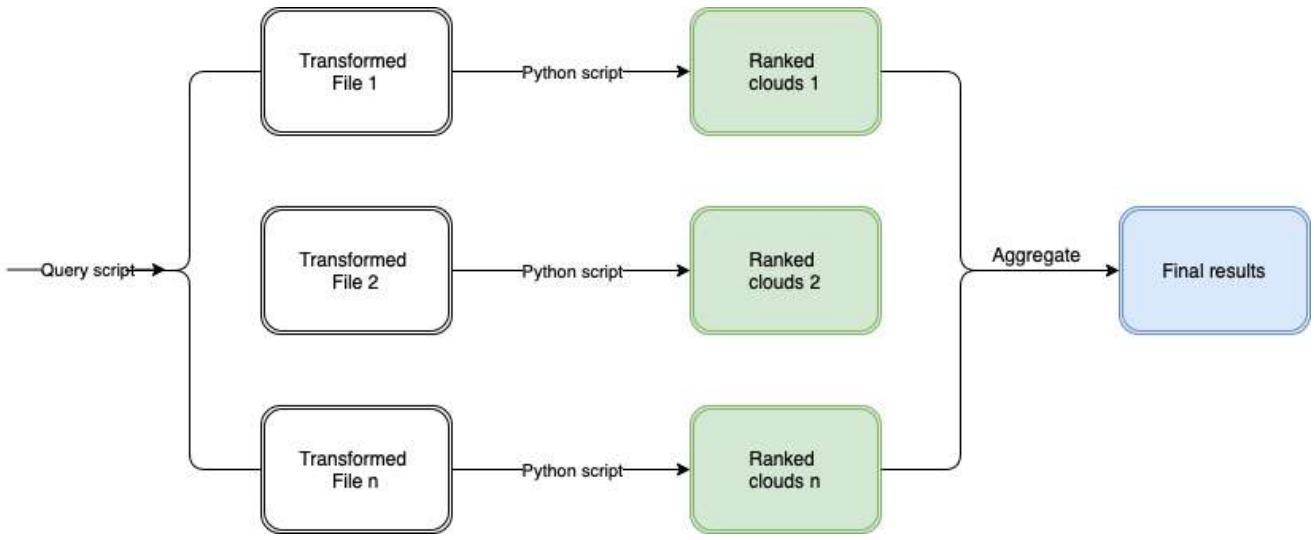

Figure 4.4: Schema of execution of the query script.

that were made during development will be documented in the following.

4.4.1. Implemented outlier metric and aggregation functions

In order to explain the implemented outlier metric, a deeper look into how Local Outlier Factor (LOF) works is in order. To do so, a bunch of definitions need to be explicated. All of them are taken from the LOF paper [**?**].

[k-distance]

For any positive integer k, the k distance of an object p, denoted as k-distance(p), is defined as the distance d(p,o) between p and an object o $\in$ D such that:

- for at least k objects o' $\in D \setminus \{p\}$ it holds that d(p,o') $\leq$ d(p,o)
- for at most k-1 objects o' $\in D \setminus \{p\}$ it holds that d(p,o') $<$ d(p,o)

[k-distance neighborhood]

Given k-distance(p), the k-distance neighborhood of p contains every object whose distance from p is not greater than the k-distance. In mathematical terms,

$N_{k-distance(p)}(p) = \{q \in D \setminus \{p\} \mid d(p,q) \leq k - distance(p)\}$

[Reachability distance]

Let k be a natural number. The reachability distance of object p is defined as

$reach - dist_k(p,o) = \max\{k - distance(o), d(p,o)\}$

Simply put, the reachability distance from p to o with respect to k is the distance between p and o, but at least the k-distance of o, as defined in definition 4.4.1.

Figure 4.5 depicts a nice example. Let's take point o as an example, with k = 3. The reachability distance between o and p2 is simply their distance, since p2 is not part of the k-neighborhood of o. But since p1 is, their reachability distance gets "pushed back" to the k-distance of o. The reason is that in so doing, the statistical fluctuations of d(p,o) for all the p's close to o can be significantly reduced.

From now on, we will refer with MinPts to the minimum number of objects that make up a cluster. With that in mind, we can define:

[Local reachability density]

The Local Reachability density of an object p is defined as:

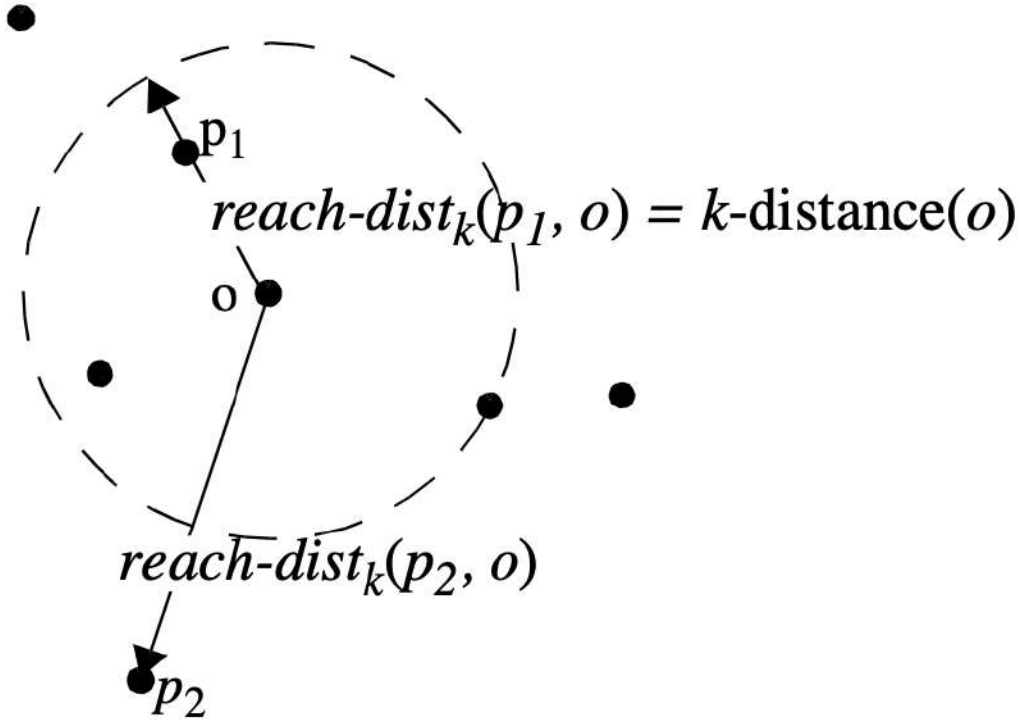

Figure 4.5: Example of reach distances from o for k = 3

$$lrd_{MinPts}(p) = \frac{1}{\frac{\sum_{o \in N_{MinPts}(p)} reach-dist_{MinPts}(p,o)}{|N_{MinPts}(p)|}}$$

Simply put, the LRD is the inverse of the average reachability distance based on the Minpts-neighborhood of p. For instance, if all the Minpts-neighbors of p are in the same exact spot of p, the local reachability distance of p will be ∞. This also entails that, if the reachability distance from p to its Minpts neighbors is higher, the local density will be - naturally - lower.

Therefore, the higher the local density of a point p, the higher it is closer or part of a well-defined cluster.

[Local Outlier Factor]

The Local Outlier Factor of p is defined as:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|}$$

So the Local Outlier Factor of p is the average of the ratio of the local reachability density of p and the local reachability density of the MinPts around it. Basically, the lower lrd(p) and the higher the lrd of its MinPts neighbors, the higher the LOF of p will be. This is in order to capture the intuition that if p's local density is much smaller than the ones of its neighbors, it's likely that p is an outlier.

In fact, the paper also demonstrates (with a sketch proof) that the LOF for any point p "deep inside" a cluster C is approximately 1, with a margin of error of ϵ. This is because, of course, the local reachability density of a point in a cluster will be very similar to the ones of nearby points.

In general, the higher the LOF, the more a point can be considered an outlier, therefore the more distant it is from a cluster of points. Back to our application, this is undesirable for our purposes, since we would want the inverse relationship: a query point should score the highest if it is very close to a cluster, so if its Outlier score is low. Of course, we could just invert the values, but we chose to instead represent the Outlier score of a query point as its **Local Reachability Density** of definition 4.4.1 instead of LOF. This score encapsulates what we are looking for much better, without comparing the density of the point to the density of its neighbors: this way we just get an idea of how close a query point is to one of the document's semantic clusters of vectors.

Summing up, the Outlier function mentioned in Algorithms 1 and 2 was implemented by using Definition 4.4.1 of Local Reachability Density from the LOF paper [**?**].

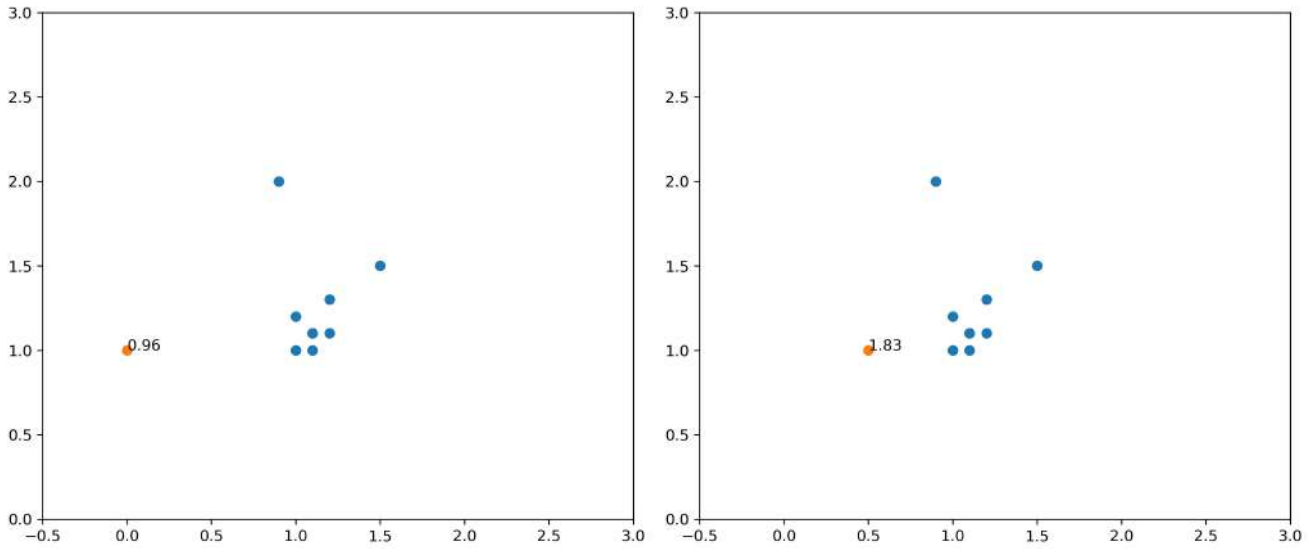

Figure 4.6: Variation of Local reachability density with k=3.

As seen in figure 4.6, the LRD as an outlier/density metrics works as expected. The closer the query point (depicted in orange) is to a cluster of vectors, the higher the value will be.

Naturally figure 4.6 was plotted in a two-dimensional plane, but the properties of this metric stay the same even in higher dimensional environments, as can be seen in 4.7 and 4.8. Of course, the vectors involved will have hundreds of dimensions, so it can be helpful to picture what this metric entails in more complex domains.

Vicinity to the cluster is not the only thing that LRD measures. A very important metric is the **density** of said cluster. As seen in Figure 4.9, the point and its neighborhood stand still, but since the cluster is less dense, its LRD score decreases.

The Local Reachability Density (LRD) metric is dependant on one parameter, k, which is the number of neighbors of the object p to be considered. In other words, it is the cardinality of p's neighborhood to consider to compute the metric. If k is a little higher than the number of tensors in a cluster, the LRD of p will plummet significantly, so it should be chosen carefully. In order to avoid this phenomenon, we will compute LRD with small values of k, but deeper studies are needed. Of course, the higher the k, the higher the confidence on the resulting LRD value, which will certainly not be misled by random noise in the dataset.

One other possibility is to compute k **dinamically**, based on the properties of the cloud of vectors at hand. This is beyond the scope of this report and will not be considered.

In order to fully implement Algorithm 1, only the aggregation functions are currently missing, since the aforementioned LRD gives an implementation for the Outlier metric and BERT is our take on the Transformer Neural Network.

The first aggregation function needs to aggregate the outlier metrics for each sentence with respect

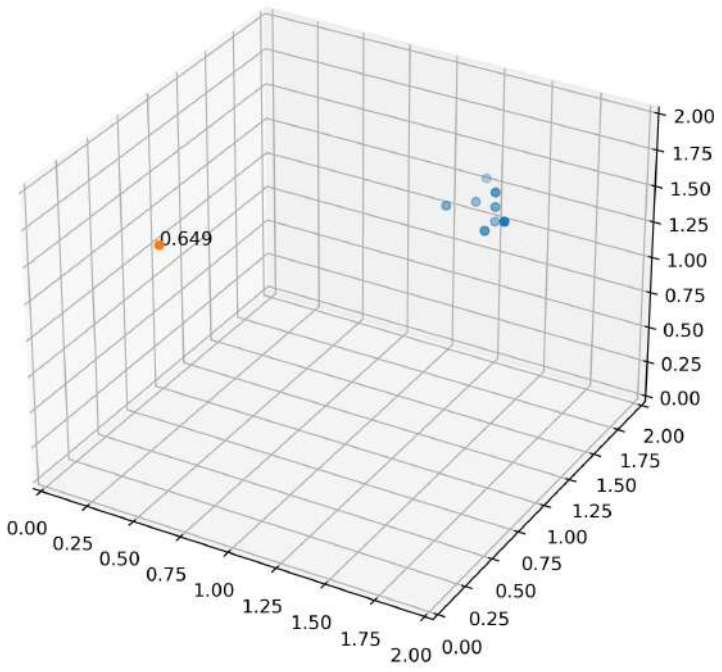

Figure 4.7: LRD in a 3d environment with k = 3 (1)

to each query term. The input will be a dictionary containing the sentence as the key and an array of LRD values, one for each query term, as value.

$$sentence : [Outlier(sentence, q_i) \dots Outlier(sentence, q_n)]$$

A good way to handle this function was just to choose the mean of the values in the array, and this is the design choice for our implementation. Therefore, the first aggregation function is just the mean of the aforementioned values.

The second aggregation function, instead, has to compute all the values for each sentence of the document. This second aggregation function is only necessary if we are computing the Outlier metric for each individual sentence, instead of the whole document at once. The second aggregation function takes as input a dictionary constructed in the following way:

$$\begin{gathered} document1 : \{sentence : score \\ sentence : score \\ \dots \\ sentence : score\} \end{gathered}$$

$$\dots$$

$$\begin{gathered} document_m : \{sentence : score \\ sentence : score \\ \dots \\ sentence : score\} \end{gathered}$$

There are, naturally, multiple ways and options to handle this dictionary and obtain a single score for each document. One way would be, for instance, to use the **mean function** once again, or to take the mean of the three highest scores.

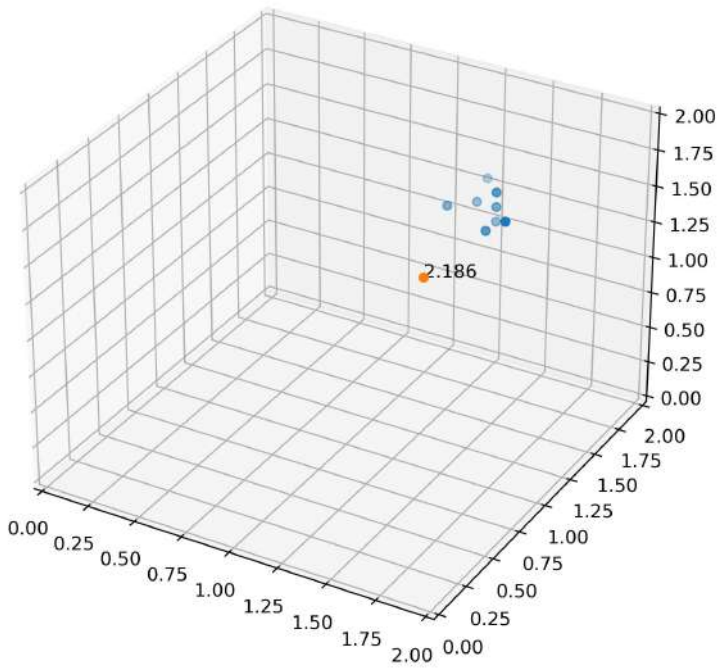

Figure 4.8: LRD in a 3d environment with k = 3 (2)

The way the second aggregation function was implemented in the system is akin to the Passage Retrieval techniques first mentioned in 2. The idea is to promote a document based on the closest sentence to the query points, which means the second aggregation function was implemented as a **max function**. This way, we are ranking the documents that contain the most relevant passage to the query terms first, as if it were a Passage retrieval task. Although, instead of returning only the sentence, the entire document is part of the ranked results.

In mathematical terms, the score associated to each document in our implemented system is:

$$Score(document, Query) = \max_{sentence \in document} \{Score(sentence, Query)\}$$

where

$$Score(sentence, Query) = \frac{\sum_{queryterm \in Query} local\_reachability\_density(queryterm)}{|Query|}$$

These are the formulas for a classic IR task. If the model has to perform a re-ranking task, there are multiple possibilities to modify the computation of the score. One of the most basic ways is to weight the score of each document for the TF-IDF score of the terms it contains. Many of the performed tests were done by using an edited version of the above formula by adding such weighting schema:

$$Score(sentence, Query) = \frac{\sum_{queryterm \in Query} tfidf(document, queryterm) * LRD(queryterm)}{|Query|}$$

The proposed cloud model is extremely versatile and customizable. Each of the design choices involved were compared with multiple other options and many more possibilities are still perfectly viable. One of the core strengths of the Cloud model described in this report is the fact that it can also be

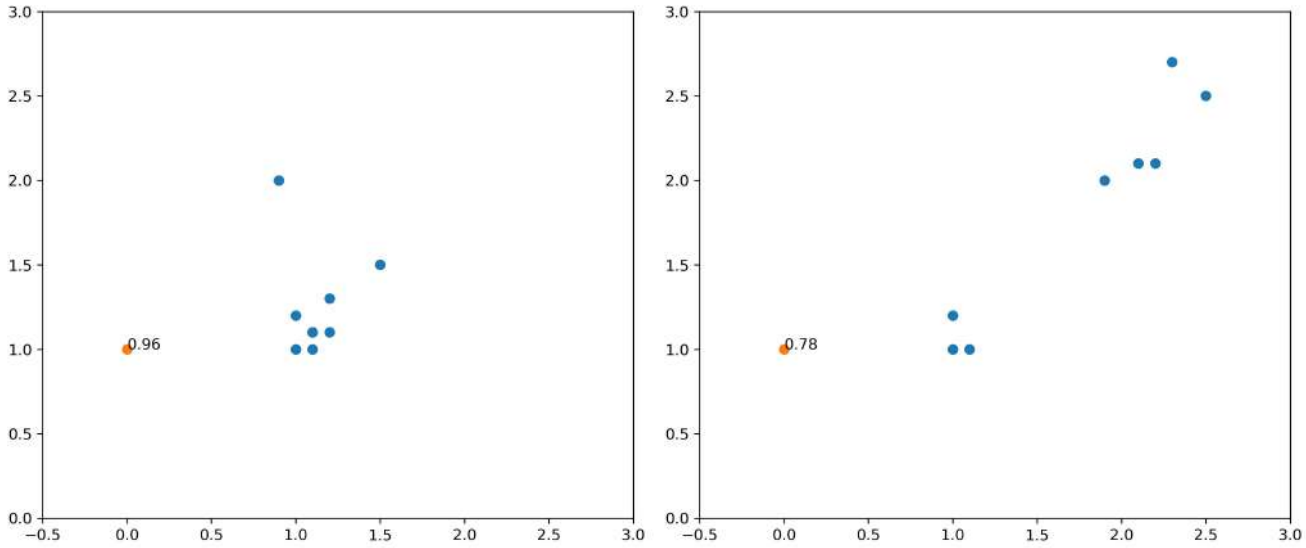

Figure 4.9: Variation of Local reachability density with k=3 due to the cluster's density.

modified for each task ad-hoc, selecting the best transformer, aggregation functions, and parameter k, for the job it needs to execute.

4.4.2. Testing manager

The **Testing manager** module helped running all the tests in a parallel way given the arguments. In particular, the necessary arguments are the following:

- The parameter k, indicating the number of neighbors for computing the Local reachability distance as seen in the previous section.
- The argument "document_wide", to decide which algorithm to use between algorithm 1 and algorithm 2.
- The argument "weighted", to decide whether or not to use the auxiliary TF-IDF weighting schema.
- The query, naturally.

Any other parameters was only used internally to coordinate tests and obtain result metrics.

The purpose of the testing manager is to ease the execution of the scripts directly into the cluster of computers, made available by the Department of Computer Science and Engineering and Mathematics DISIM, University of L'Aquila. The cluster, nicknamed Caliban, makes use of CentOS 7, and the script was built in a way to optimize the great computational resources it offers, including many processing cores and GPUs.

The testing manager module also includes the functions necessary for Reranking. In particular, this task follows a different structure since the scores are individually saved in a different folder than the original bm25 scores, allowing for quick re-weighting of the reranking score c.

4.5. Graphical User Interface

A basic GUI was also built to ease early testing and have a visual representation of the clouds of vectors with respect to the query tensors. The GUI was built with the library PythonSimpleGUI and was never intended for public use, thus the simplistic look.

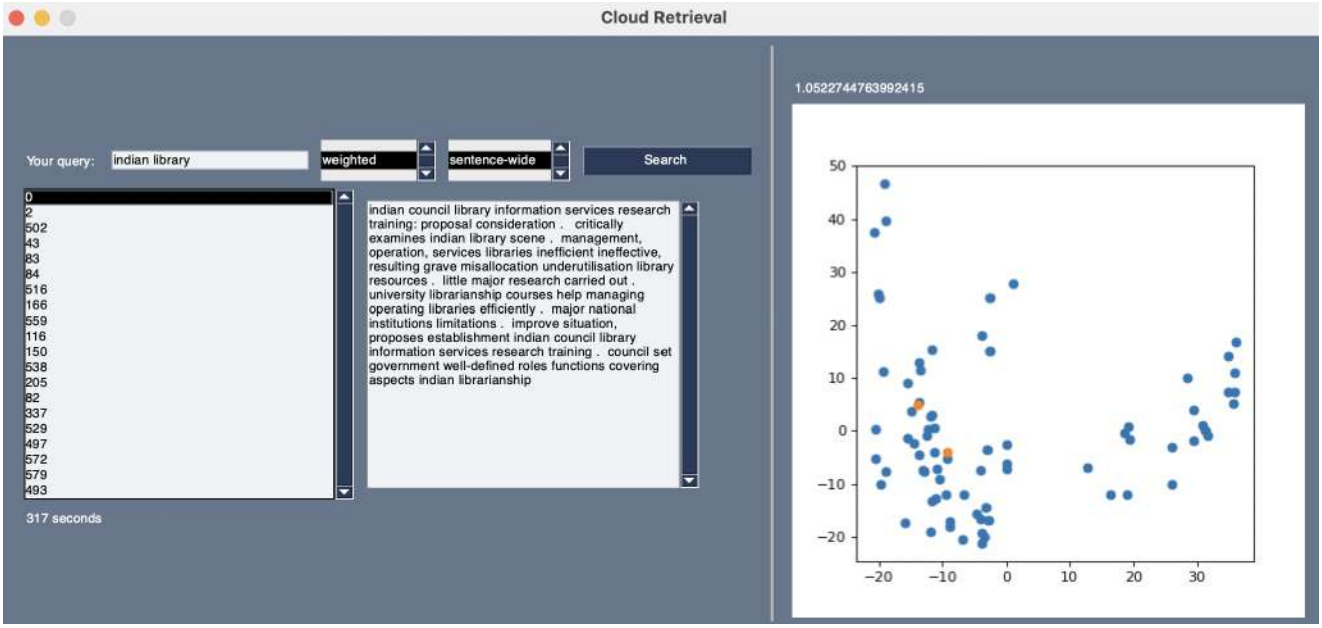

Figure 4.10: Screenshot taken from the GUI.

The GUI allows to query for anything in the collection, while setting the type of algorithm that should be used, either weighted/not weighted or sentence-wide/document-wide. Once the results are given, the user can click on one of them to visualize on the right the text contained in that document, the visual representation of that document's cloud of vectors, and the score with respect to the query.

4.5.1. Image Composer

In order to visualize the clouds in the GUI, a module called Image Composer was built. The module takes as input the clouds of vectors and the transformed query, and uses Principal Component Analysis (PCA) to reduce the dimensionality of the vectors down to two in order to scatter plot them into a two-dimensional plane. The orange points, as depicted in figure 4.10, are the query points. The blue ones represent, of course, the cloud of vectors of the selected document.

4.6. Environment

All experiments were run on the department's cluster of computers on a custom Miniconda environment.

4.6.1. Miniconda environment

The Linux release of Miniconda had to be manually installed in the cluster in order to run these experiments. The chosen language for implementation was Python, to simplify the programming phase and make use of a number of useful libraries dedicated to Transformers and Neural Networks. The custom conda environment was built from the ground up to include the following main libraries, listed in the table below, in alphabetical order:

hugging-face	Popular API for models, transformers and tools. Provides easy methods to call and use the BERT Model and BERT tokenizer, trec_eval, and more.
numpy	Standard library for math tools and vectors management. One of the core building blocks of the implemented model.

pandas	Standard library for big datasets management. Used in this project to handle the raw dataset, preprocess its contents, and prepare the files including the clouds of vectors.
PySimpleGUI	Library used for a basic implementation of User Interface.
scipy	Useful library that includes numpy. Used to ease the computation of the Outlier metric (Local reachability distance)
torch	Python library to manage tensors. Very useful to handle the results of the BERT Neural Network.
transformers	API From the hugging-face library that specifically includes the BERT model and all of the methods to interact with it.

4.6.2. Shell command

Since the experiments were all launched in a CentOS environment, the best way to allow the OS to manage and check job statuses via the **qsub** and **qstat** commands was to execute the code inside Bash scripts. To this aim, a general .sh file was built with the goal to be as flexible as possible, and launch all possible experiments.

The resulting BASH script activates the conda environment and takes as inputs the same parameters that the script needs to be executed, but from command line. It processes them and passes them to the actual python script that starts the query. The results were then saved locally in the cluster, downloaded via an ssh connection, and processed and visualized locally.

5 Model evaluation

In this section we will present results for all the experiments that were carried out to evaluate the proposed model's performances. We will do so in an organized, schematized way in order not to lose any information.

First of all, we will briefly detail our Cloud Retrieval model and the parameters it needs to be executed. We will then present the datasets onto which the tests were carried on and, lastly, specify how the results and evaluations will be organized in this chapter.

The Cloud Retrieval model, as shown in sections 3 and 4, requires one main parameter k to function. Simply put, k is the size of the neighborhood of a tensor that we will consider to compute its Local Reachability Density, the score used by the proposed model. It is also worth noting that two algorithms were presented in section 3, but only the first one, the Sentence wide Algorithm 1, was used. While the Document wide Algorithm 2 certainly holds some merit, time constraints made it impossible to test it properly, so while some early data was extracted, it was not enough to draw some reasonable conclusions.

The Cloud Retrieval model will be tested on two main Information Retrieval tasks, both of which were detailed in section 2: Ranked Retrieval and Re-ranking. For the latter task, another parameter will be needed: α will represent the weight given to the Cloud Retrieval score, while (1-α) will multiply the original score, basically the score of the original collection to be reranked. More details will be given later, but the main grasp is that we will be using the `sentence wide Algorithm, that needs a parameter k for Ranked Retrieval and parameters k and` α `for Re-ranking`.

Keeping this in mind, we can now briefly present the datasets that will be used for our testing.

5.1. Datasets

Hereafter we present all the datasets we are going to employ in the evaluation of the models.

5.1.1. LISA

The dataset that was mainly used during our testing is called LISA. LISA is a general, multi-topic collection of 6000 documents with themes ranging from technical specification of computers to mundane topics like libraries. In total there are 26228 sentences in the collection. The LISA dataset was actually split into numerous files that had to be unified into a single pandas dataframe, and formatted according to the aforementioned schema to enable further processing.

LISA's Qrels are comprised of 35 queries, each annotated with a number of hand-picked relevant results, ranging from only 2 documents for certain queries, up to 20 for others.

5.1.2. MS Marco

The second dataset used to gather data is MS Marco [**?**]. MS Marco stands for Microsoft MAchine Reading COmprehension and is a collection of datasets that focuses on search tasks using deep learning. The datasets can be used for various purposes including Natural Language Generation, Passage Ranking, Crawling, etc...

MS Marco is incredibly popular and has been the de facto standard for Information Retrieval systems for a while. However, the Cloud Retrieval model is extremely expensive for computational resources and the goal of this report was to test out its results and compare it with different baselines, not optimize its performance, although great work has been done in that direction as well (as shown in 4).

Therefore, for time constraint reasons, using the entire MS Marco dataset, comprised of almost 9 million passages, was not feasible. The experiments noted in section 5 were carried on on a sampling of the MS Marco dataset as a result. The sampling was done in the following way:

- The qrels of the MS Marco dataset were analyzed and ALL the documents listed as Relevant were collected. This ensures that no matter the query, all the relevant documents are present in the sampled dataset.
- Another 5000 randomly sampled documents were added to dilute the dataset.

The resulting sampled dataset was composed of around 20.000 documents, more than double those of the entire LISA dataset.

For the same reason, 40 out of the provided queries for MS Marco were sampled. In particular, the first 40 queries of the small version of the dataset provided by the Python library ir_datasets were selected and saved.

MS Marco is a much bigger and sensible dataset, so testing the Cloud Retrieval model on its entirety might be very helpful to test out which queries are answered better and which ones are poorly responded to. We can assume that longer, more contextualized queries work better because BERT manages to understand them wholly. However, a full scale test would be needed to draw some clear-cut conclusions.

Naturally, MS Marco and its queries underwent the same pre-processing pipeline as LISA, composed of multiple steps. The end result is a dataset of almost 20.000 documents and 40 queries all preprocessed to have no stopwords, all lower case, and without non-ASCII symbols.

5.2. Preliminary experiments

A wide range of experiments were carried on to evaluate the proposed model. The main effort of this section is to schematize and visualize them such that the results will be easily understood and the implications intuitive. For this reason, we will start by detailing the experiments that were carried on in order to choose which Transformer Neural Network to use (candidates were BERT and Word2Vec) and a good parameter k. Once these decisions are set in stone, all the remaining experiments will be shown and divided into two categories: Ranked Retrieval and Re-ranking.

5.2.1. Determining the best Transformer Neural Network

The main candidates for the role of Transformer Neural Network for the implementation of Algorithm 1 were Google's BERT and Word2Vec. While they both share the same goal, e.g., transforming tokens into semantic-embedded tensors which are geometrically closer if the words are semantically related, they significantly differ both on dimensionality (BERT's tensors are up to 8 times bigger than Word2Vec's default settings) and the ability to perceive and embed context sensitivity inside the tensors. As a matter of fact, Word2Vec always embeds the same tokens to equal tensors, while BERT analyzes the context around the token to generate an unique tensor every time that token is transformed.

In order to determine which is best, the Ranked Retrieval task was executed on the `LISA dataset` with k=3 and k=4 using both Neural Networks.

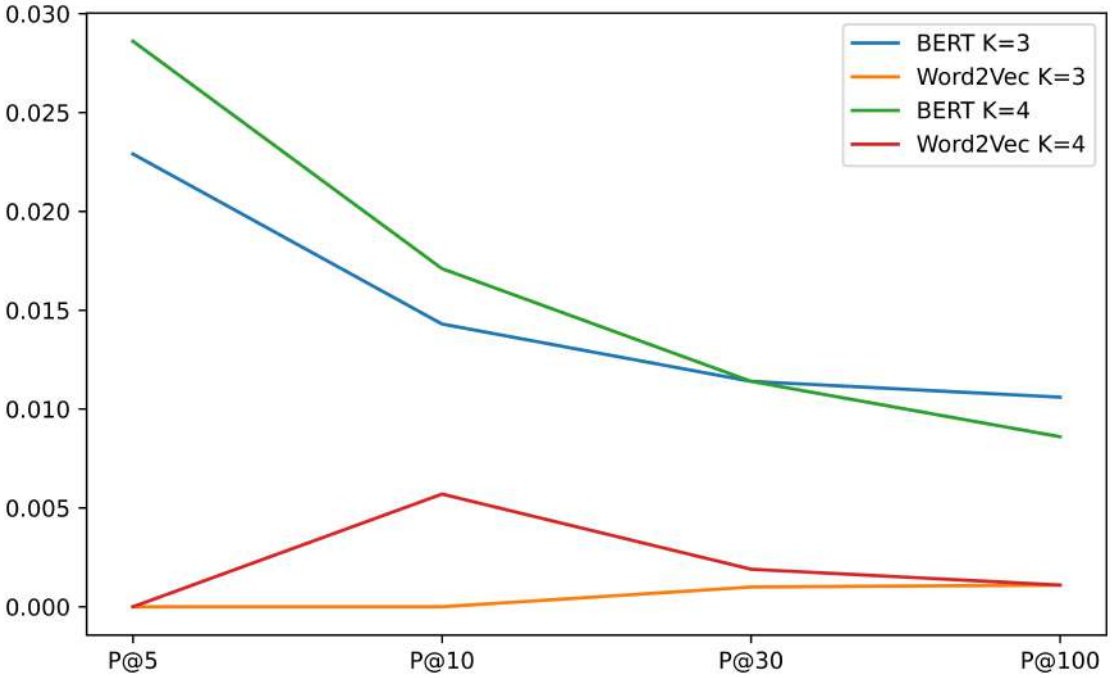

Figure 5.1: Precision at different values for the Transformers comparison.

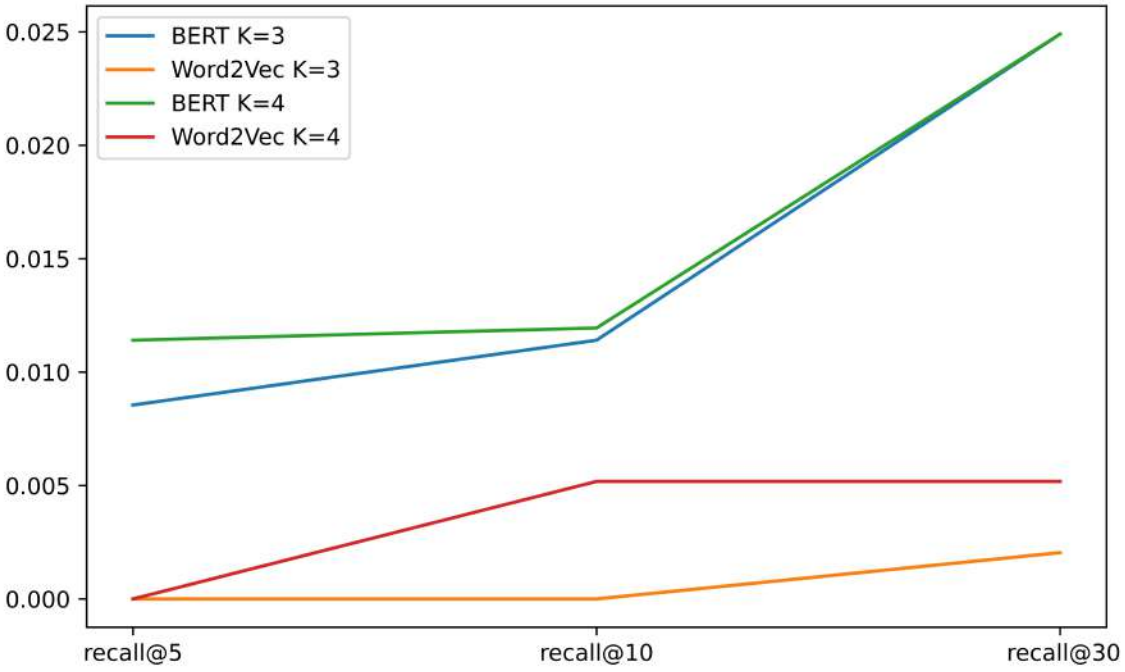

Figure 5.2: Recall at different values for the Transformers comparison.

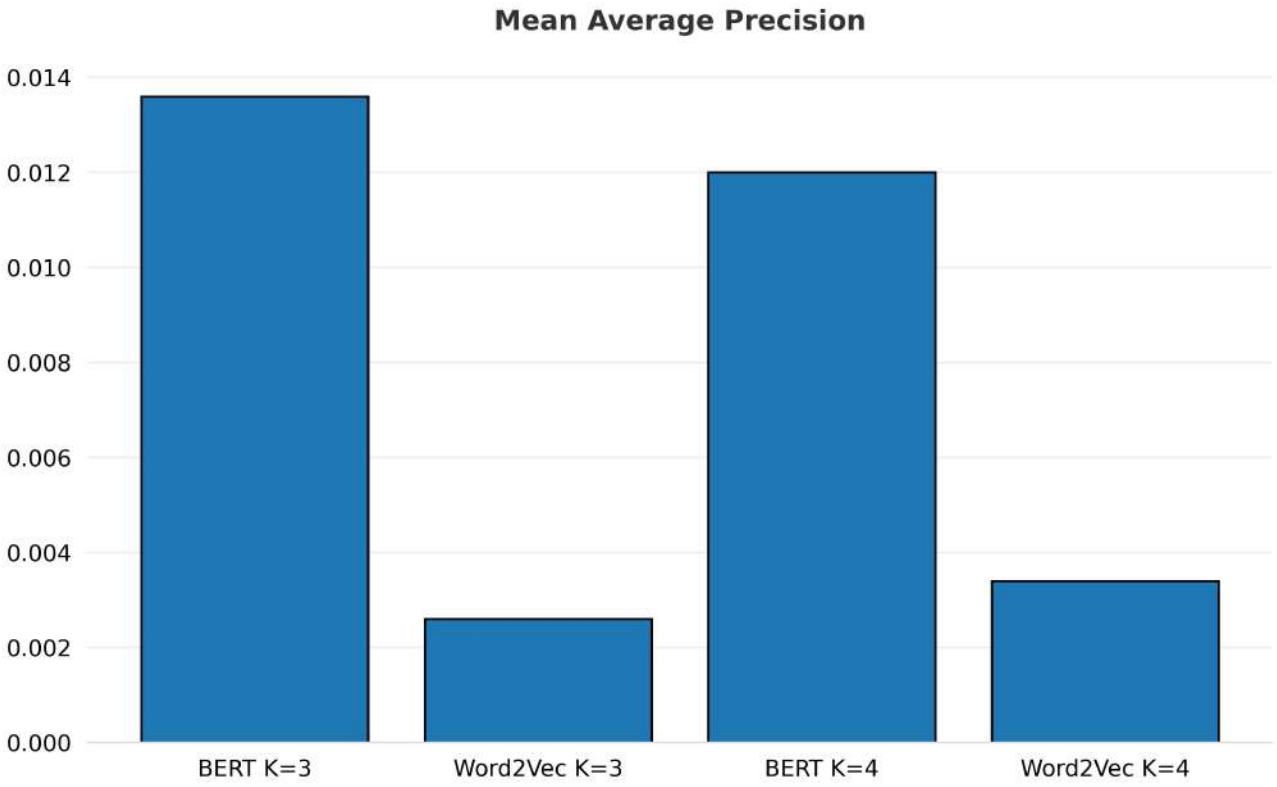

Figure 5.3: map for the Transformers comparison.

Figures 5.1, 5.2 and 5.3 show the gathered metrics for this comparison. It is easy to see that BERT outperforms Word2Vec for every single metric and the comparison is not even close. Upon further inspection, a clear problem arises: Word2Vec encodes the same tokens with the same tensors, regardless of the context around it. It does so in a way that transform words that are semantically close to tensors that are geometrically close. While that is a desirable trait for our purposes, it hides a much bigger problem for our density metric.

Imagine that a document contains the same tokens multiple times, which is not at all a far-fetched assumption. Since Word2Vec transform all of those tokens to the same tensors, the resulting Cloud of Vectors will contain a super-condensed cluster of tensors with distance 0 between them. This quickly results to an abnormal density measure for the query terms that are near these clusters, since the Local Reachability Density for them will be 0, flattening the scores and making them mostly meaningless. One obvious solution would be to remove duplicate tokens for each document, but we would therefore lose information on how important that token is, or how often it appears, failing to reward query tokens that are near terms that often appear in the document. We will not study the problem further, but it is important to keep in mind that this is one possible path to pursue.

Given this rationale and the results supporting it, we will be only using BERT's tensors for future computations. Therefore, all the following results, unless explicitly noted otherwise, will have been obtained with the sentence wide Algorithm implemented using BERT.

5.2.2. Choosing the best parameter k

In order to choose the best parameter k, more experiments were carried on on the `LISA dataset`. More specifically, metrics for the Ranked Retrieval task were gathered by using k = 3 and k =4.

For starters, we assume that a smaller k is generally better for the proposed model. That is because we want a transformed query token to "link" with a given cluster, and if the size of k even slightly exceeds the cardinality of said cluster, the Local Reachability Density will, by definition, be lowered considerably,

which is not a desirable outcome. For more details, take a look at section 3.

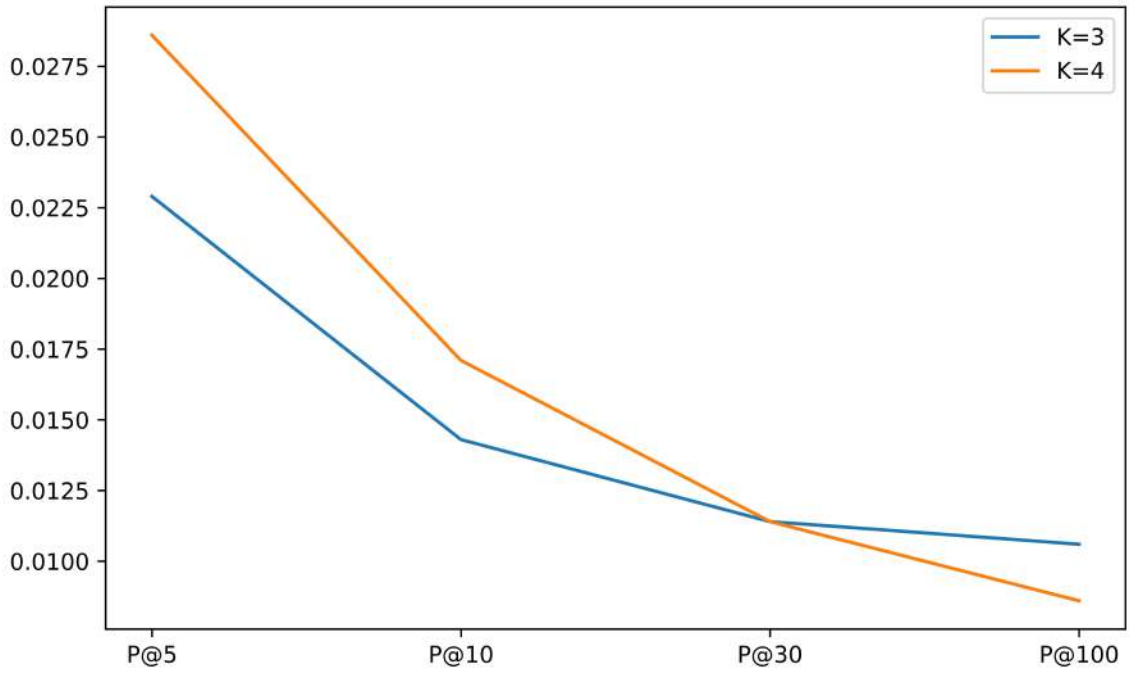

Figure 5.4: Precision at different values for variation of parameter K.

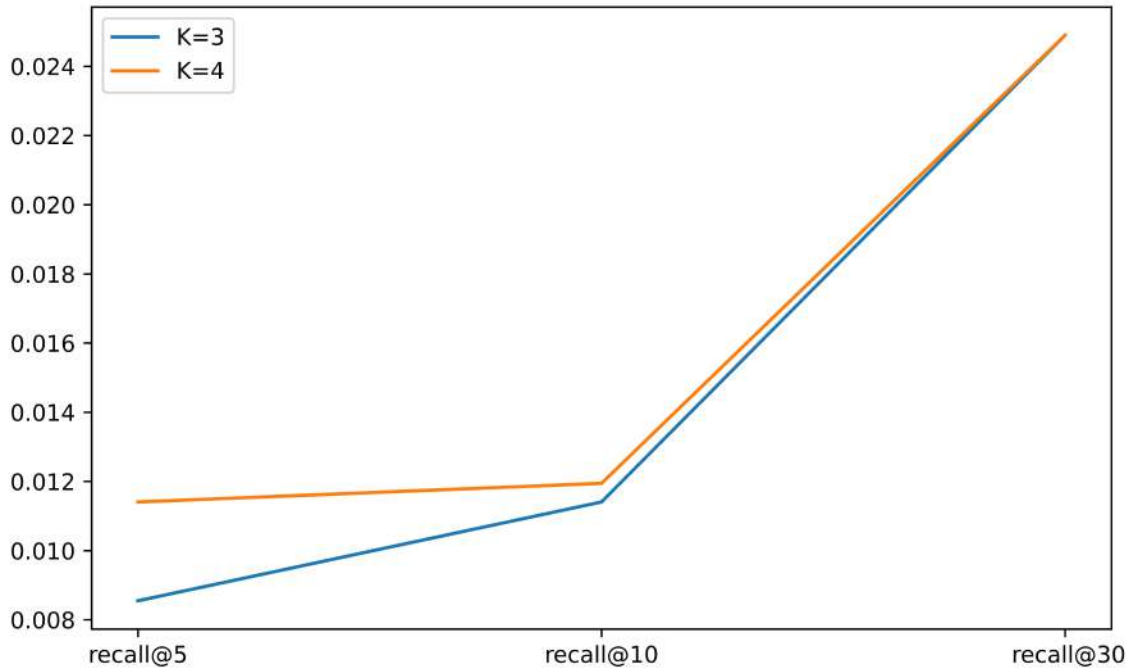

Figure 5.5: Recall at different values for variation of parameter K.

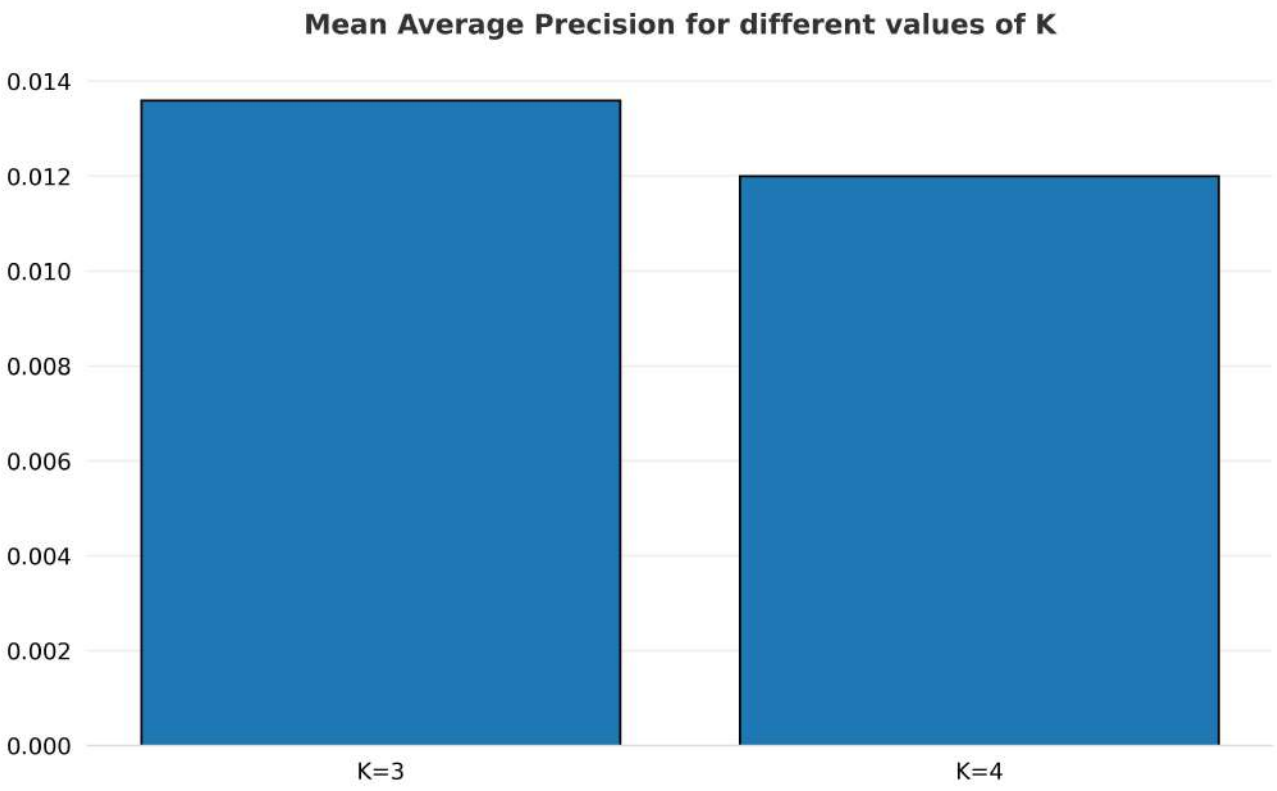

Figure 5.6: map for variation of parameter K during the Ranked Retrieval task.

Figures 5.4, 5.5 and 5.6 show that the algorithm for a slight variation of k behaves in a mostly similar way, with K=4 edging out on early precision and K=3 being a little more consistent when the retrieved documents are many more.

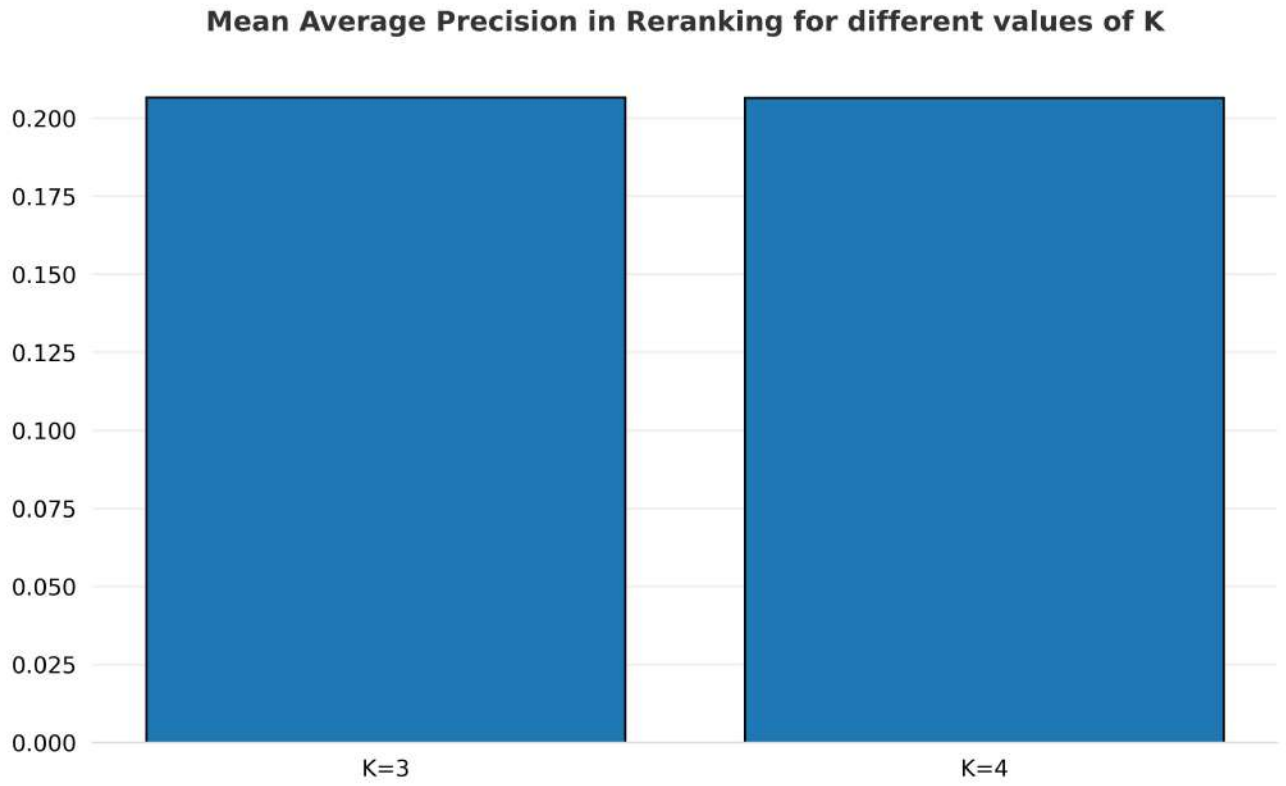

Figure 5.7: map for variation of parameter K during the Reranking task.

In order to have even more information, a Reranking task was executed on the `LISA dataset` with k=3 and k=4. As seen in figure 5.7, k=3 has slightly higher map even in the reranking phase, even though the results are almost identical. All experiments shown from here onwards will therefore use k=3, but the research for an optimal value of k is still open, but shall not be investigated further in this report work.

5.2.3. Overview of the Experiments

As mentioned before, the first experiments shown were aimed to show the rationale that brought us to the design decisions of the transformer neural network and parameter k to use for the actual comparison. These are intended to be a sort of preamble to the actual experiments, that are listed in table 5.8

TASK	DESCRIPTION	DATASET	GATHERED METRICS	GRAPHICAL REPRESENTATION
RANKED RETRIEVAL	Comparing metrics between multiple versions of Cloud Retrieval, with variable k	LISA	Precision at 5,10, 30, 100 Recall at 5,10,30,100	Lineplots
	Comparing metrics between the Mean of Embeddings baseline and Cloud Retrieval model with k = 3		Precision at 5,10, 30, 100 Recall at 5,10,30,100	Lineplots
			Mean Average Precision	Barplot
		MS MARCO	Precision at 5,10, 30, 100 Recall at 5,10,30,100	Lineplots
			Mean Average Precision	Barplot
RERANKING	Comparing performances of BM25 and Cloud Retrieval Reranking with k = 3 and variable α	LISA	Precision at 30, MAP	Lineplot, barplot
	Comparing performances of BM25 and Cloud Retrieval Reranking with k = 3, α = 0.75		Precision at 5,10, 30, 100 Recall at 5,10,30,100	Lineplots
			Mean Average Precision	Barplot
		MS MARCO	Precision at 5,10, 30, 100 Recall at 5,10,30,100	Lineplots
			Mean Average Precision	Barplot

Figure 5.8: Overview of the experiments.

This section will strictly follow figure 5.8, so that a reference is always available.
Summing up, all future experiments will be using:

- Sentence-wide algorithm 1
- BERT Neural Network
- k=3

With this information in mind, we can present the experiments that will be illustrated over the course of this chapter. As for table 5.8, we will be dividing the experiments in two main tasks. The goal of each of them is the same, obtaining the best possible results for any given query. While the process is different, the evaluation function (provided by trec_eval) is the same, and the metrics of interest will be identical.

5.2.4. Baselines

Ranked Retrieval tests will compare the proposed Cloud Retrieval model to the Mean of Embeddings (MoE) baseline presented in section 2. This baseline represents the most basic approach to an embedding-based Ranked Retrieval technique, and is therefore a good reference to make sure that the model makes sense.

Our goal with the Re-ranking tasks, instead, is to compare the results with the original ranking list obtained with BM25 (details in section 2). For this task we will be using Algorithm 3 and evaluate whether or not Cloud Retrieval manages to improve bm25's results on both reference datasets.

A specific effort was made to visualize results in a consistent way throughout the section. For ease of understanding, the various values of Precision and Recall (at different steps) will always be plotted as LinePlots, while the Mean Average Precision differences will be visualized as barplots. Any other metric can be fetched from the table placed at the end of the section.

5.3. Evaluations in the Ranked Retrieval task

The gathered results for Algorithm 1 with BERT and k=3, resulting in a Ranked collection of documents, will now be shown in comparison to the Mean of Embeddings baseline. Results for the `LISA dataset` will be displayed first, followed by those obtained with the sampling of the `Ms Marco dataset`. At the end of this section, a table containing all metrics for all tests will be attached for transparency of results.

We will start by visualizing the results obtained on the LISA dataset, as seen in Figures 5.9, 5.10 and 5.11.

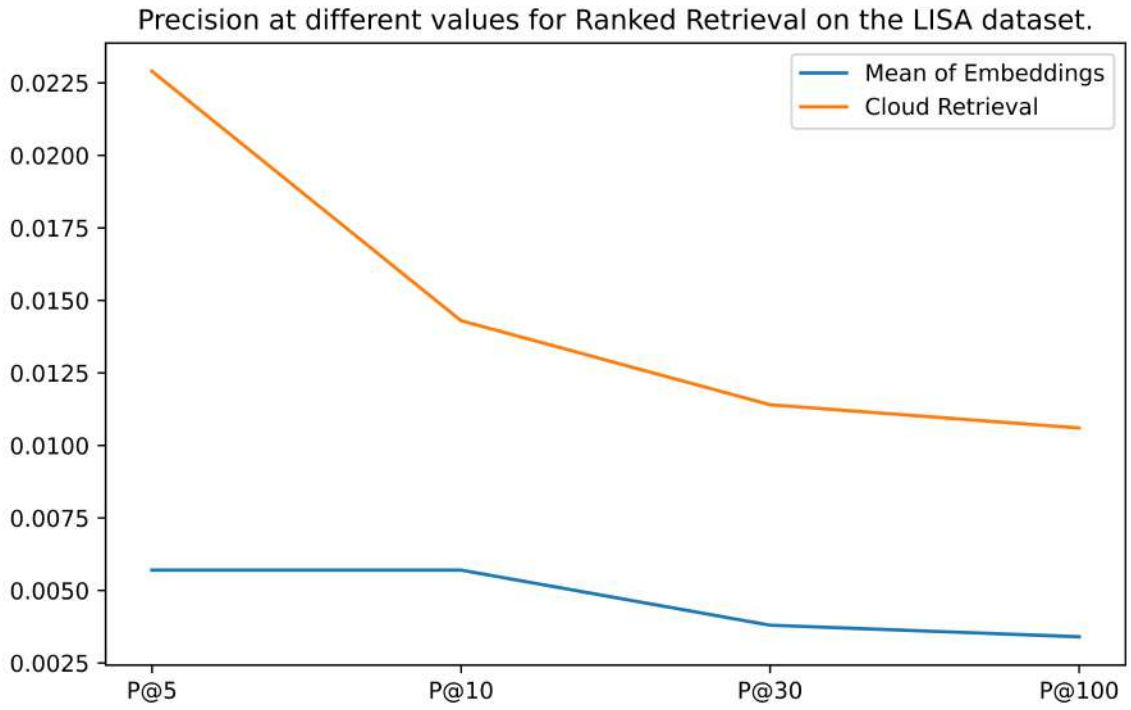

Figure 5.9: Precision at different values for Ranked Retrieval on LISA.

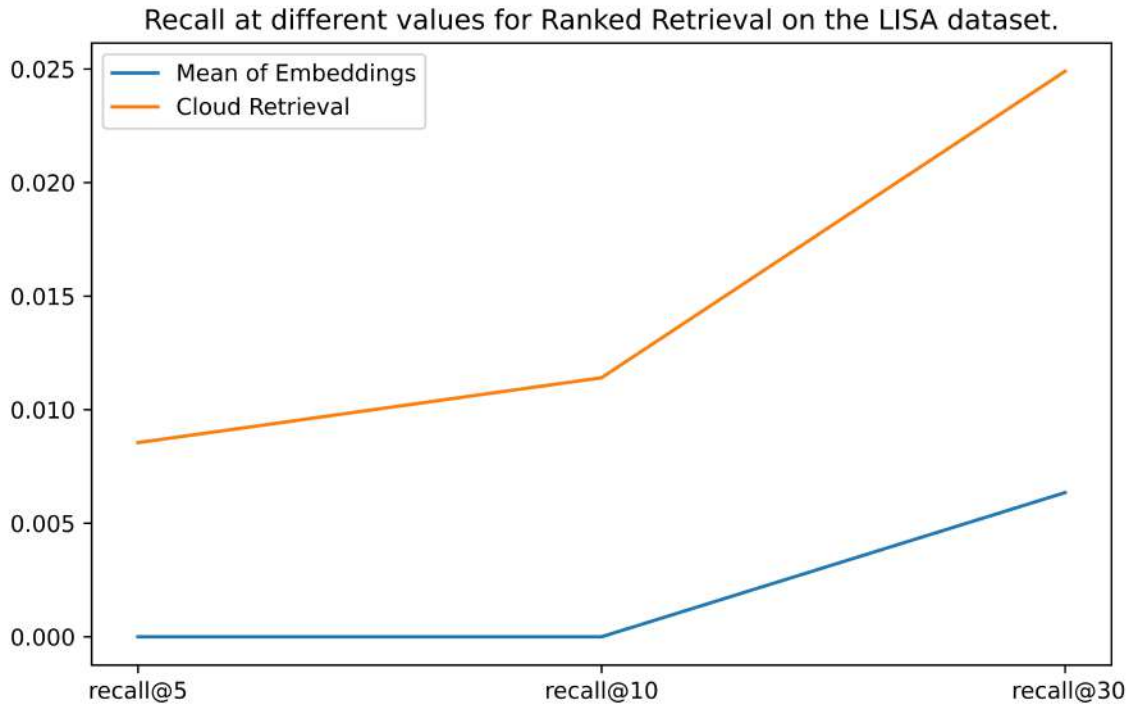

Figure 5.10: Recall at different values for Ranked Retrieval on LISA.

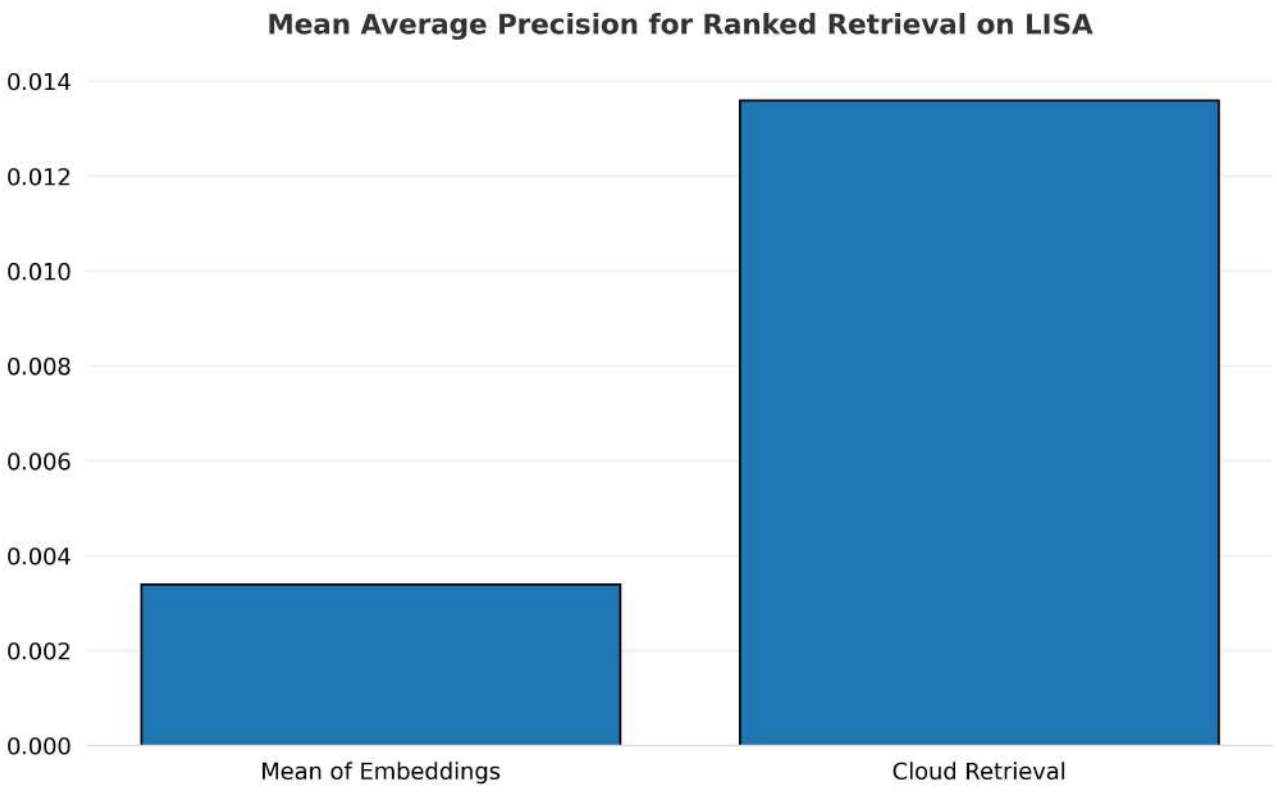

Figure 5.11: Mean Average Precision for Ranked Retrieval on LISA.

We will now show the same metrics obtained on the MS Marco dataset before analyzing these values.

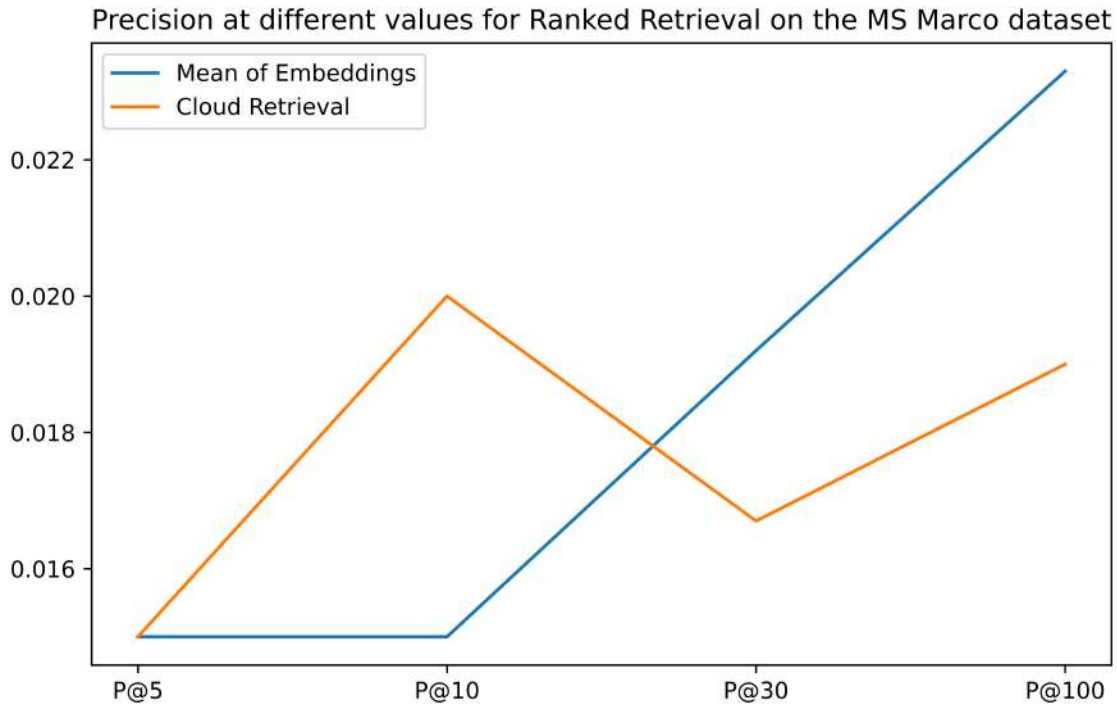

Figure 5.12: Precision at different values for Ranked Retrieval on MS Marco.

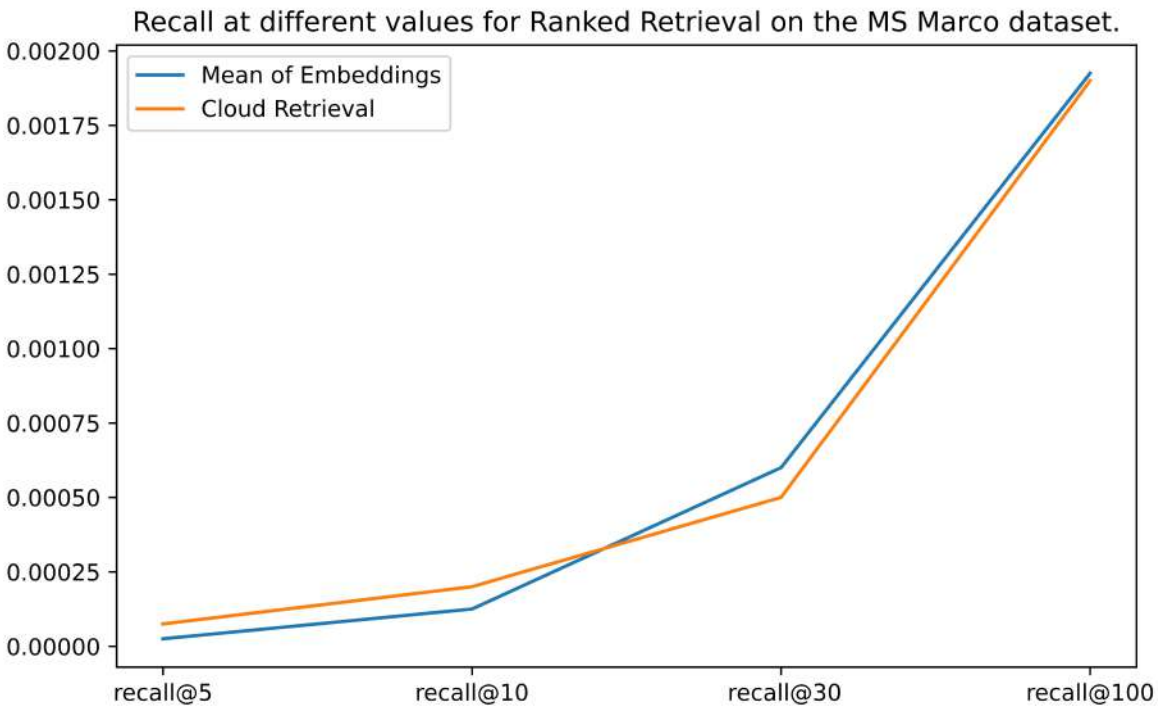

Figure 5.13: Recall at different values for Ranked Retrieval on MS Marco.

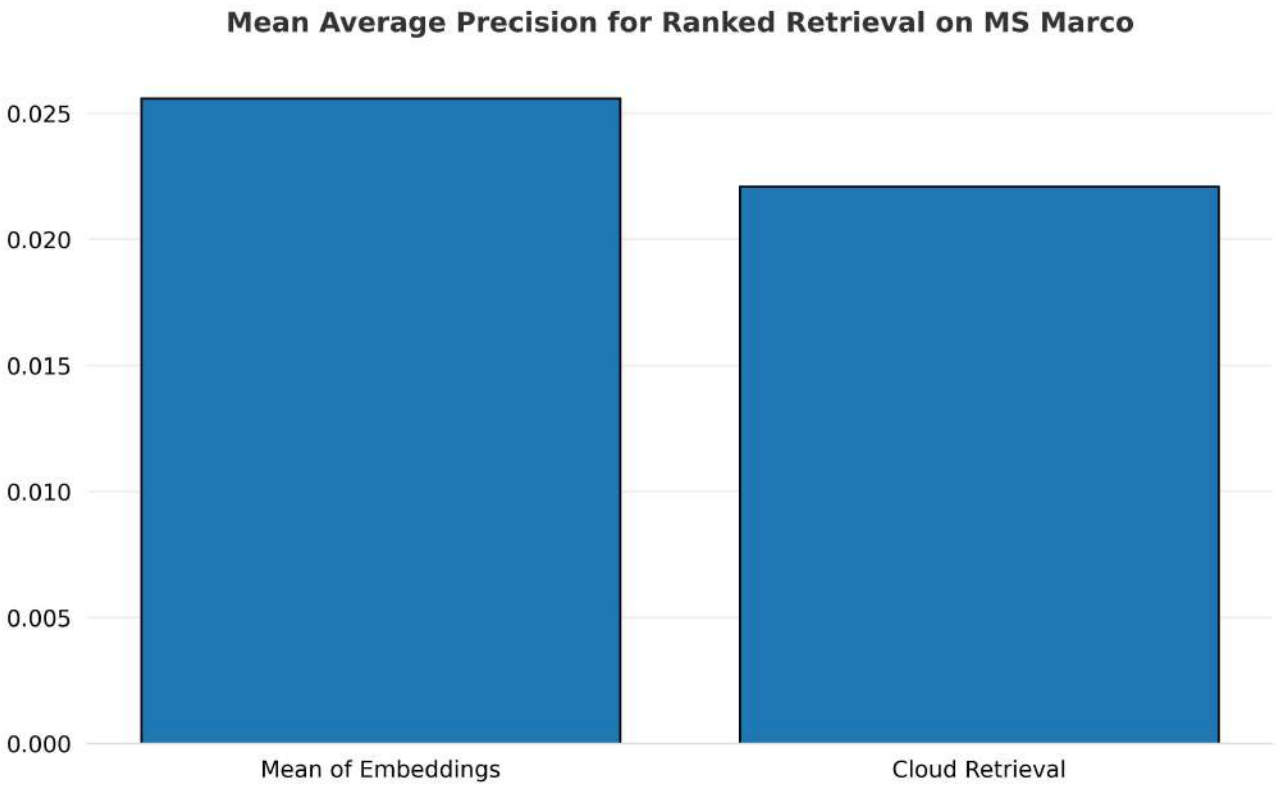

Figure 5.14: Mean Average Precision for Ranked Retrieval on MS Marco.

The patterns that can be observed from these results are twofold: for the LISA Dataset, it is clear that the BERT version of Cloud Retrieval, while outperforming the MOE baseline, still doesn't achieve great results. The LISA dataset is certainly not easy to work with for some of its inherent properties, such as very long and detailed queries and very few relevant documents associated with them. Even standard tools like BM25 do not shine when working with LISA, as we will see with later results. Nonetheless, the precision is extremely low and those difficulties should not be used as a way to cope and redirect some of the model's problems. It still stands that the Cloud Retrieval model comfortably surpasses the baseline and therefore certainly holds some merit and semantic significance when the queries are so detailed that BERT can fully understand their context.

Looking at MS Marco's results, they are not so stellar and interpretation is not easy. It is clear that the model is not able to keep up for some of the metrics, and the reason might be due to the shorter queries. As a matter of fact, of the model's problems was found early in development: since queries are transformed by BERT, which take as input the entire sentence to try and transform the tokens into semantic-embedded tensors, the entire context should be contained within that one string. However, queries, especially in a Web Search environment, are rather small, and often reduced to one or two keywords. Due to this reason, we observed that BERT had a lot of trouble placing related tensors close to each other. One way to mitigate this problem is to repeat the first term of the query as the last one. Since BERT, as mentioned in section 2 takes information from both directions starting from a specific token, the first one suffers from a lack of vital details about what comes beforehand. Copying it as last helps adding some context beforehand while still placing it correctly as the last token. While this does help mitigate the problem, it is likely still not enough. Possible solutions include techniques such as Query expansion, where we aim to gather as much information as possible regarding the query before starting the lookup on the Cloud Retrieval system. To this aim, we developed a very simple algorithm to fetch a wider context given a short query.

Algorithm 4: Query expansion algorithm

```
1.  Read input query.
2.  Look for relevant documents in the collection with respect to the
 query with BM25.
3.  Gather the top n results. 4.  Concatenate the most relevant
 sentences from the top n results and transform them into a Cloud of
 Vectors CV.
5.  Start a search in the Cloud Retrieval Model using CV as the query.
```

While such an algorithm was not implemented and related results not gathered yet, it is easy to foresee how it can considerably help semantic-based retrieval given the assumption that semantically related tokens are transformed into vectors that are close to each other.

Overall, MS Marco's results appear to be worse than those obtained with the LISA dataset. This had to be expected when considering that the ms_marco dataset comprises of smaller, more direct queries, which heavily advantage a tf-idf based system, much like the classical Retrieval system. For this reason, the metrics obtained for the Reranking tasks will probably be more interesting.

5.3.1. Inserting a tf-idf weighting schema in the model

These experiments were carried on to test other parameters during the Ranked Retrieval task, but are not part of the comparisons since they were not fully tested and developed. Both of the experiments listed here were tested on the `LISA dataset`.

Figure 5.15 shows the comparison for Ranked Retrieval, with the same parameters, between the two Algorithms detailed in section 3, Algorithm 1 and 2. The latter seems to be doing a bit better, but the difference seems to be negligible for the time being. The document wide algorithm is extremely computational expensive and needs to be optimized before large scale testing is even feasible. Figure

5.16 shows the Precision results of an earlier version of the Cloud Retrieval Model that weighted the scores using a TF-IDF Matrix, being basically an hybrid of the standard model and the reranking model used on the BM25 results. Unsurprisingly, it doesn't perform as good as BM25 by itself, but the drop in precision is less steep for higher ranges of documents. This shows us that a TF-IDF score is still extremely profitable for direct queries and Retrieval. This model was later replaced by the Algorithm that performs reranking on the BM25 top results.

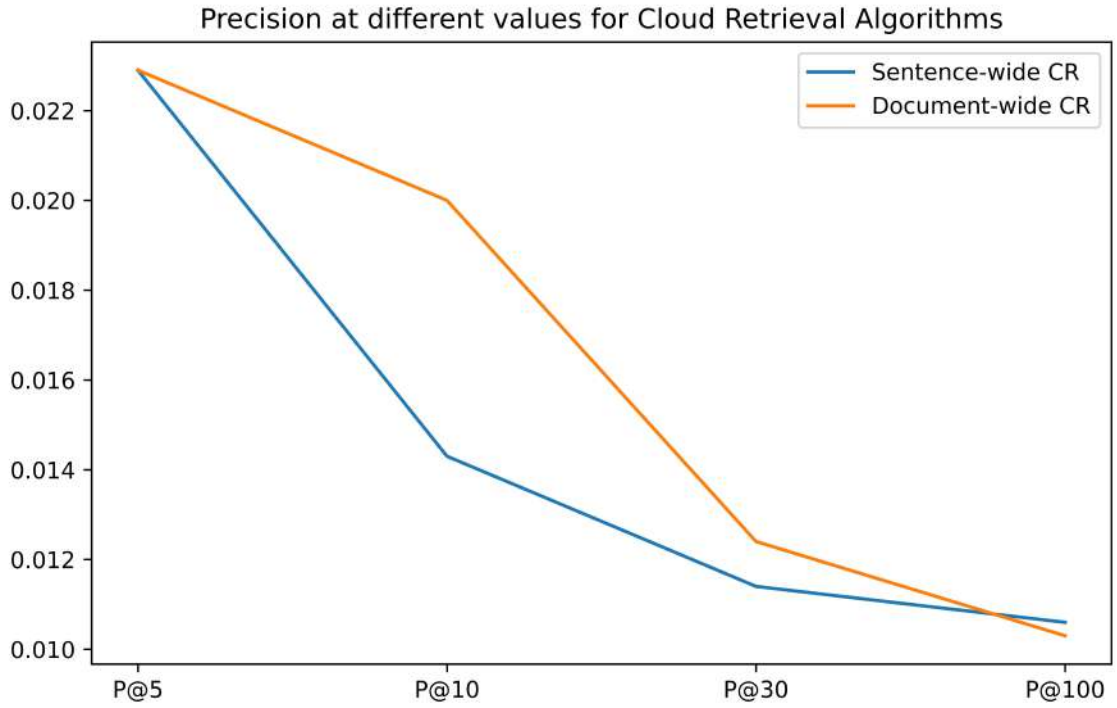

Figure 5.15: Precision at different values for Cloud Retrieval Algorithms.

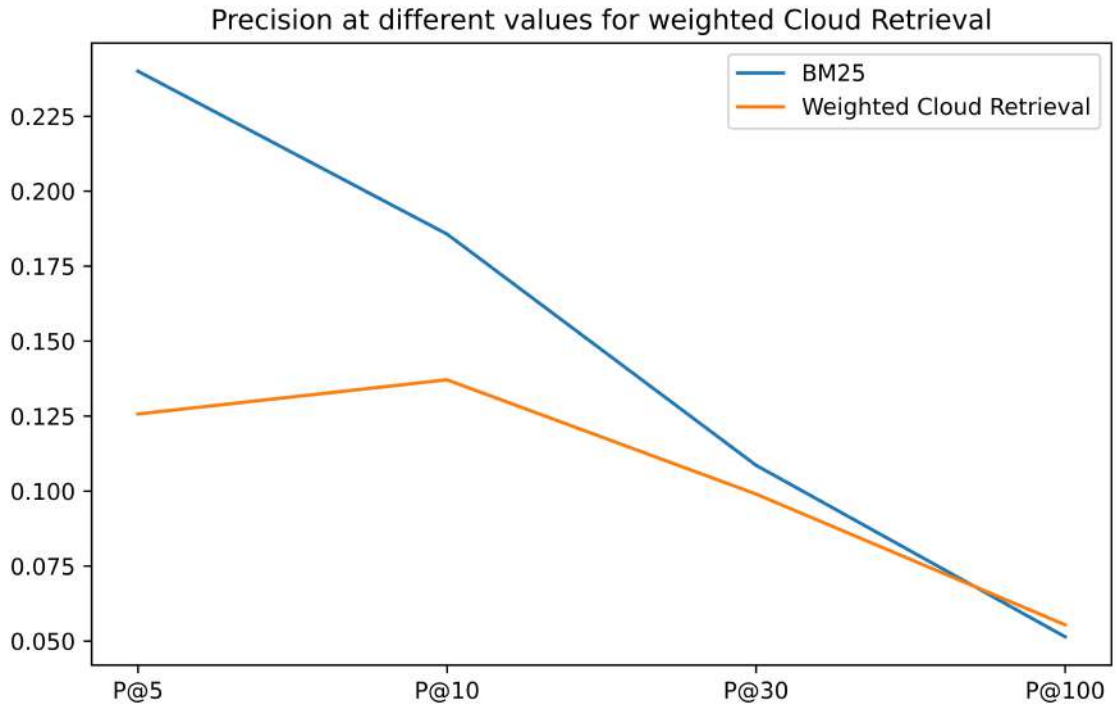

Figure 5.16: Precision at different values for Weighted Cloud Retrieval.

The next section, as for table 5.8, will detail all the experiments for the Reranking task.

5.4. Evaluations in the Reranking task

Pinning the chosen Neural Network and parameter k was enough to carry out experiments for the Ranked Retrieval task. However, when dealing with Reranking, we need one more parameter: α. Referencing algorithm 3, α is the weight assigned to the reranking score with respect to the original score.

5.4.1. Choosing the best weight

Recall that the Reranking formula is the following:

$$\text{Newscore(d)} = (1-\alpha) * \text{OriginalScore(d)} + \alpha * \text{Outlier(d,Q)}$$

So naturally, if α = 0, we are basically considering the original score of BM25, without any change whatsoever. The greater the α, the greater the impact of the Reranking score.

In order to pin the right value of α, several reranking jobs were run and analyzed, with different values for the parameter.

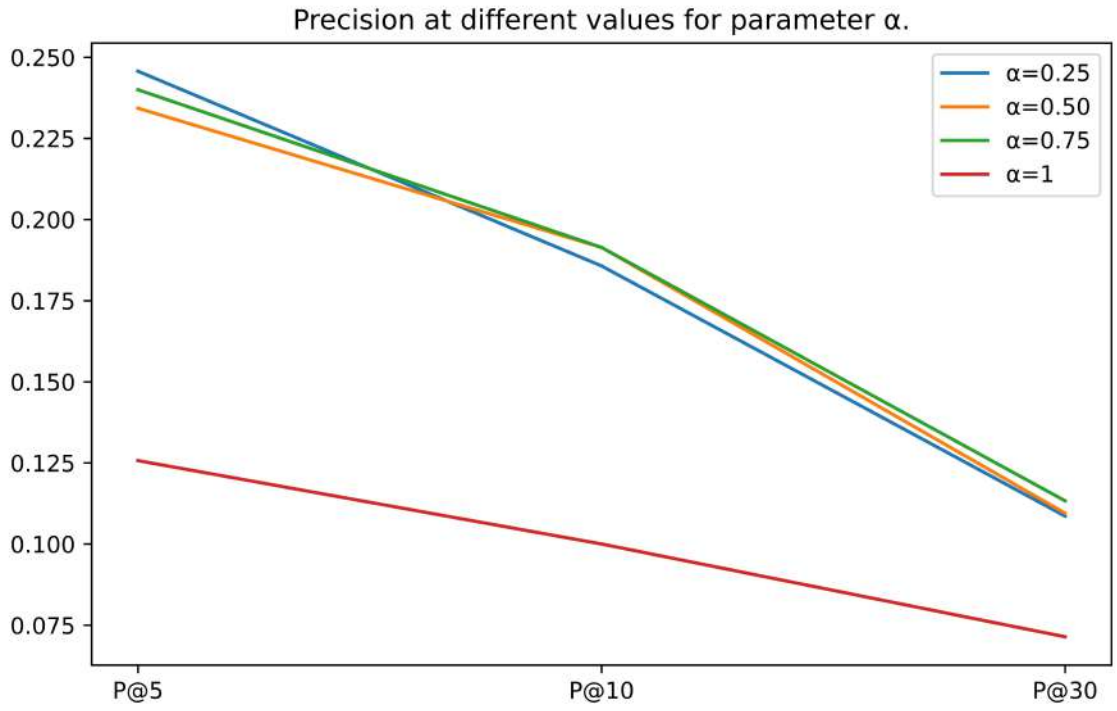

Figure 5.17: Precision for different values of α

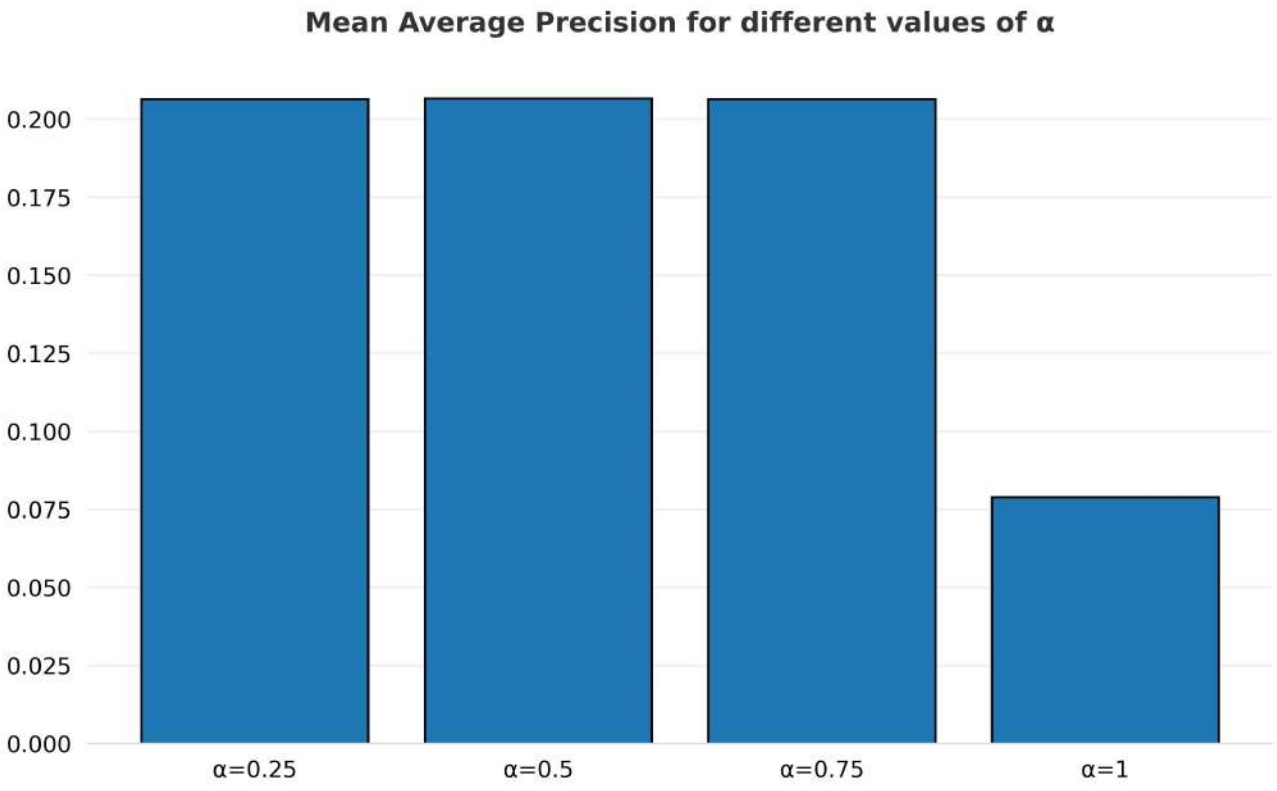

Figure 5.18: Mean Average Precision for different values of α

Figures 5.17 and 5.18 show the resulting metrics for these experiments.

When α = 1 we see a sharp drop in performance: we already established in the previous section that some sort of TF-IDF weighting, that BM25 provides, is incredibly useful to obtain good results. When α = 1, we are only relying on the pure Cloud Retrieval Model, which does not have any weighting at all.

The other values of α are pretty similar result-wise. We ended up choosing α = 0.75 for our tests because it provided slightly more consistency across larger result sets. Therefore, other than the parameters already set before (BERT, Sentence-wide algorithm, k=3) we now also pin α = 0.75 as our to-go value for the reranking weight.

5.4.2. Comparison of BM25 and reranked results

Algorithm 3 was implemented with the aforementioned parameters and experiments carried out on both LISA and MS Marco.

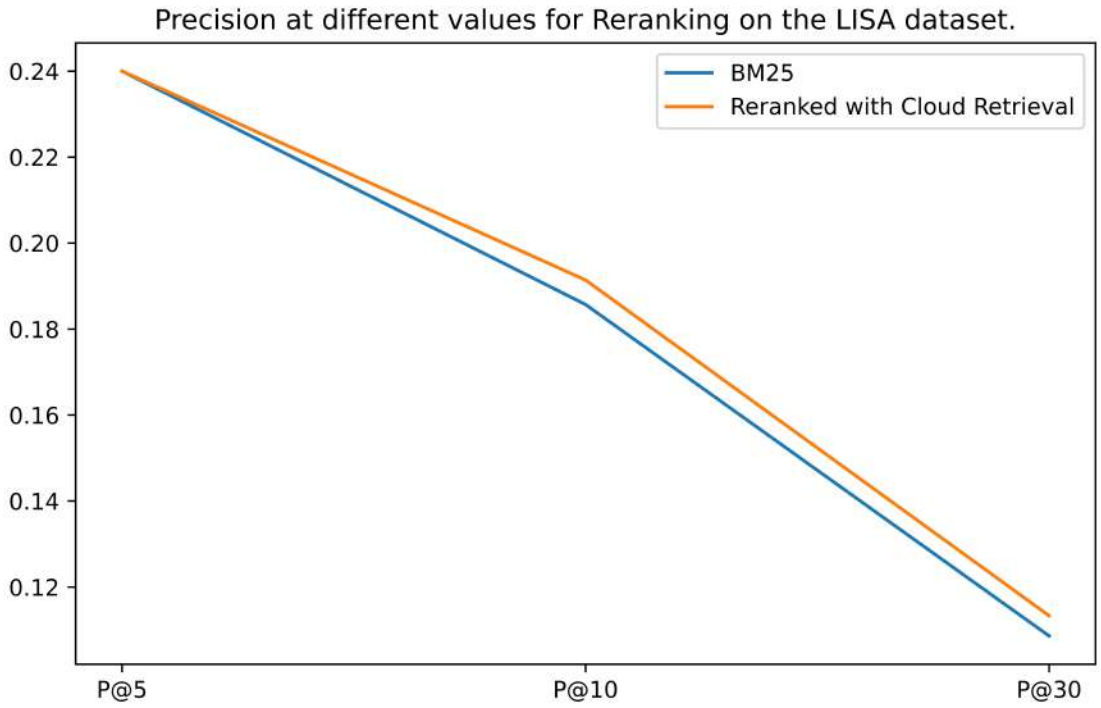

Figure 5.19: Precision at different values for Reranking in the LISA dataset.

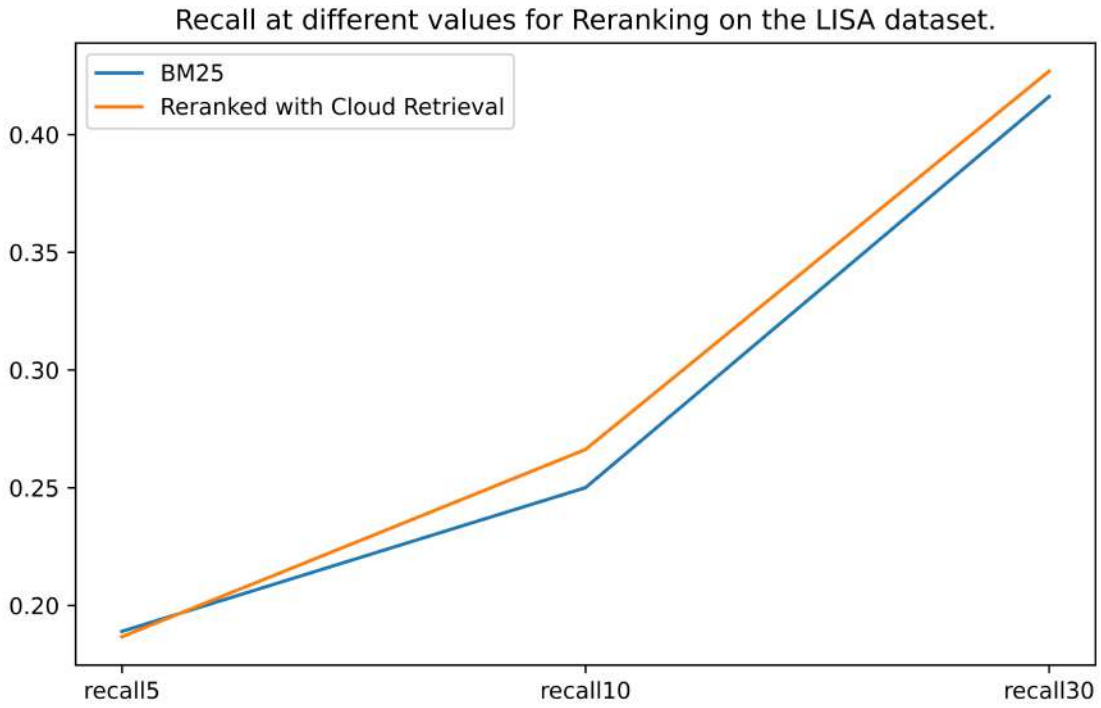

Figure 5.20: Recall at different values for Reranking in the LISA dataset.

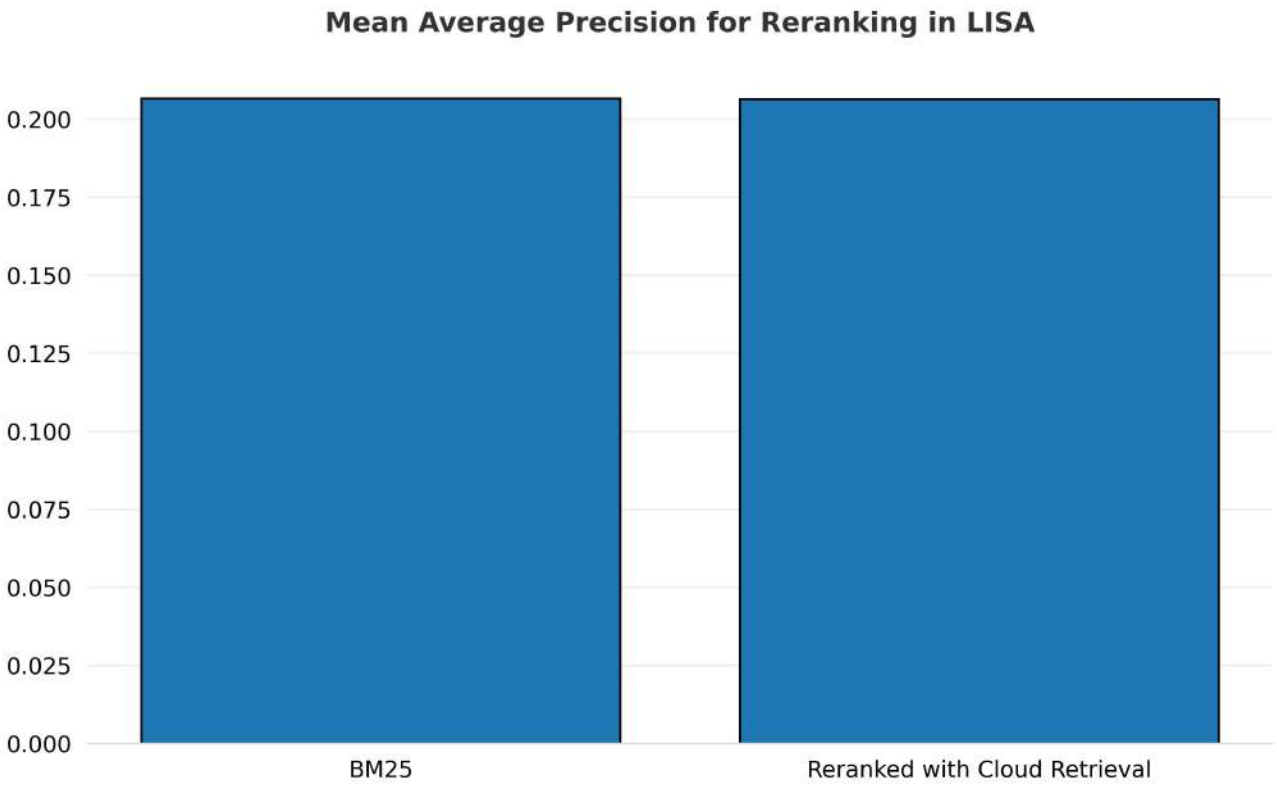

Figure 5.21: Mean Average Precision for Reranking in the LISA dataset.

Figures 5.19, 5.20 and 5.21 show how the Cloud Retrieval reranking is able to improve, albeit slightly, every single recorded metric for the given results on the LISA dataset.

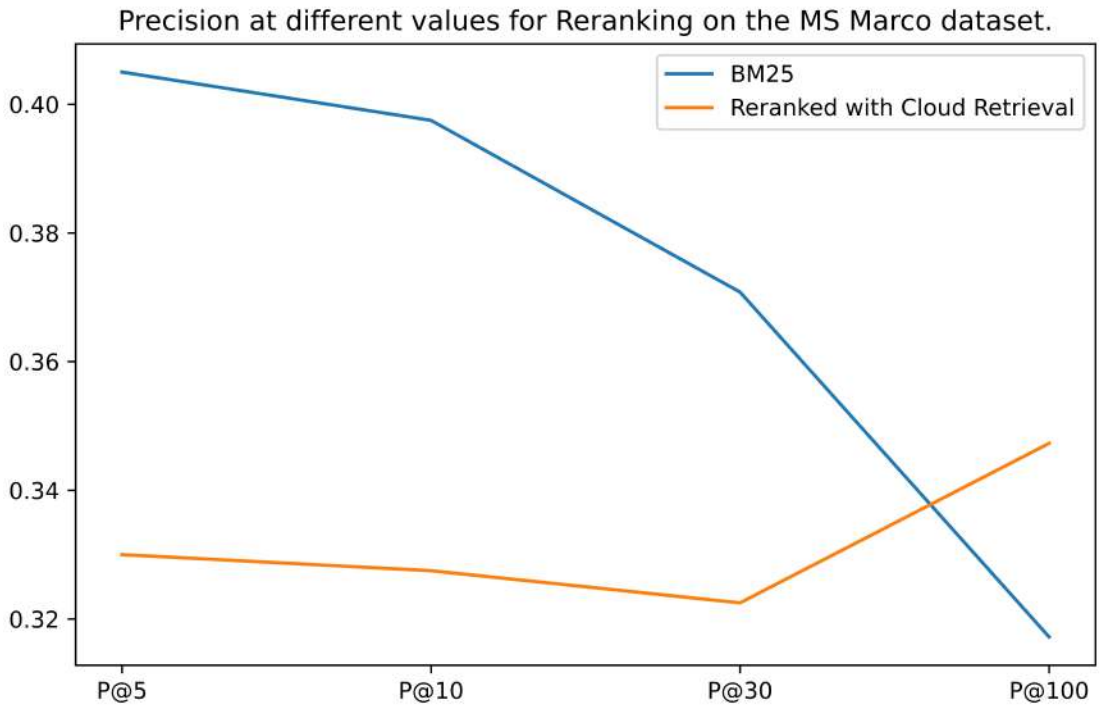

Figure 5.22: Precision at different values for Reranking in the MS Marco dataset.

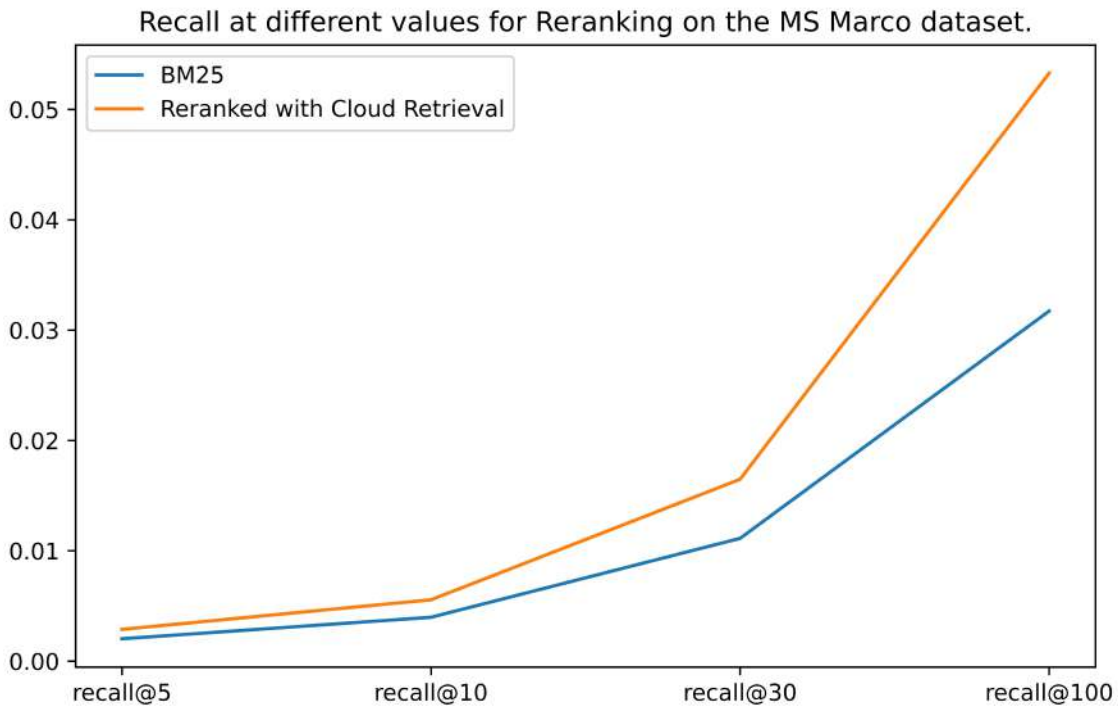

Figure 5.23: Recall at different values for Reranking in the MS Marco dataset.

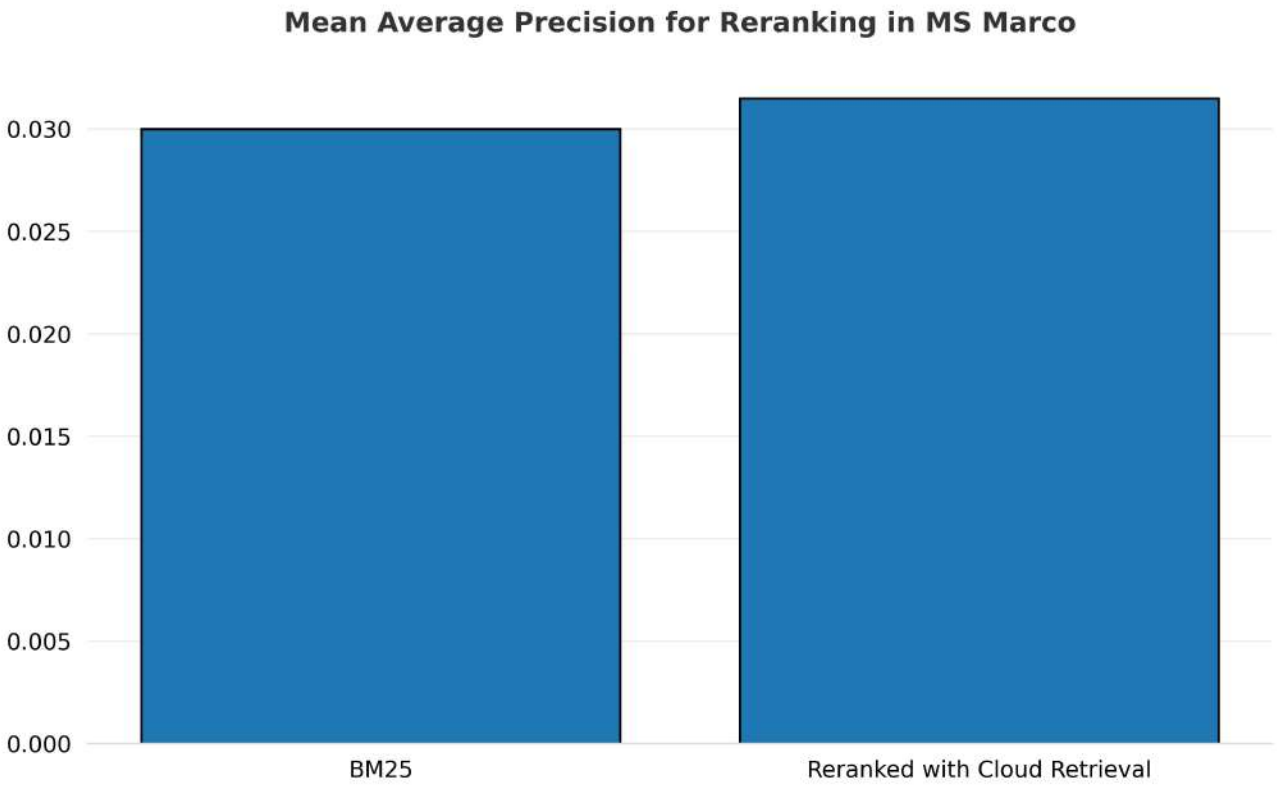

Figure 5.24: Mean Average Precision for Reranking in the MS Marco dataset.

Results for the MS Marco dataset are instead shown in Figures 5.22, 5.23 and 5.24. It is evident that there is a sharp drop in Precision using the proposed model for the first 10 documents. However, as the range of documents widens, the Reranked set of results appears to be more consistent, until it surpasses the starting precision for 100 documents. This is even more clear when looking at the recall measures. The Reranked metrics for recall are consistently better than the starting ones, with the gap widening as the range gets higher.

Lastly, the Reranked set of results also have a slightly higher Mean Average Precision.

The following is the table containing all the gathered metrics for the LISA dataset:

	label	map	gm_map	p5	p10	p30	p100	recall5	recall10	recall30
0	bm25	0.206700	0.056000	0.240000	0.185700	0.108600	0.051400	0.188936	0.250013	0.416173
1	Mean of embeddings	0.003400	0.002100	0.005700	0.005700	0.003800	0.003400	0.000000	0.000000	0.006349
2	Weighted Cloud Retrieval	0.138700	0.062800	0.125700	0.137100	0.099000	0.055400	0.115416	0.177048	0.316457
3	BERT k=3 sentence wide	0.013600	0.004800	0.022900	0.014300	0.011400	0.010600	0.008548	0.011405	0.024899
4	BERT k=3 document wide	0.015800	0.004600	0.022900	0.020000	0.012400	0.010300	0.019156	0.022381	0.063348
5	Word2Vec sw k=3	0.002600	0.001800	0.000000	0.000000	0.001000	0.001100	0.000000	0.000000	0.002041
6	BERT k=4 sentence wide	0.012000	0.004400	0.028600	0.017100	0.011400	0.008600	0.011405	0.011944	0.024899
7	Word2Vec sw k=4	0.003400	0.001800	0.000000	0.005700	0.001900	0.001100	0.000000	0.005181	0.005181
8	Word2Vec dw k=3	0.002300	0.001700	0.000000	0.000000	0.001000	0.001100	0.000000	0.000000	0.001099
9	Word2Vec dw k=4	0.002900	0.001900	0.000000	0.002900	0.001900	0.001400	0.000000	0.000000	0.003140
10	reranked k=3 sum	0.206700	0.056200	0.234300	0.191400	0.109500	0.051400	0.185399	0.253476	0.417761
11	reranked k=4 sum	0.206600	0.561000	0.234300	0.191400	0.109500	0.051400	0.185399	0.253476	0.417761
12	reranked k=3 doubled	0.207600	0.056500	0.234300	0.185700	0.110500	0.051400	0.185399	0.250570	0.419348
13	reranked k=3 tripled	0.206500	0.056500	0.240000	0.191400	0.113300	0.051400	0.186641	0.266216	0.426940
14	reranked k=3 times10	0.209300	0.055300	0.245700	0.188600	0.115200	0.051400	0.182277	0.257570	0.429644
15	reranked k=3 times100	0.177500	0.041100	0.177100	0.148600	0.099000	0.051400	0.135668	0.197221	0.333317
16	reranked k=2 tripled	0.206400	0.056300	0.240000	0.194300	0.113300	0.051400	0.186641	0.267458	0.426940
17	reranked k=10 tripled	0.205600	0.056100	0.234300	0.191400	0.110500	0.051400	0.185399	0.267453	0.419348
18	reranked k=3 tripled MEAN	0.208800	0.056900	0.228600	0.194300	0.112400	0.051400	0.183358	0.267762	0.425244

While the following is the table containing all metrics gathered for the ms marco dataset:

	label	map	gm_map	p5	p10	p30	p100	recall5	recall10	recall30	recall100
0	bm25	0.0300	0.0010	0.4050	0.3975	0.3708	0.3172	0.002025	0.003975	0.011125	0.031725
1	bm25 11 queries	0.0807	0.0728	0.9091	0.9182	0.9364	0.8336	0.000000	0.000000	0.000000	0.000000
2	bm25 25 queries	0.0469	0.0040	0.5680	0.5560	0.5480	0.4876	0.000000	0.000000	0.000000	0.000000
3	reranking k=3 s_w tripled 11 queries	0.0877	0.0778	0.8909	0.8909	0.8818	0.9236	0.000000	0.000000	0.000000	0.000000
4	reranking k=3 s_w tripled 25 queries	0.0500	0.0023	0.5120	0.5080	0.5040	0.5328	0.000000	0.000000	0.000000	0.000000
5	reranking k=3 s_w tripled	0.0315	0.0005	0.3300	0.3275	0.3225	0.3473	0.002880	0.005560	0.016480	0.053280
6	Mean of Embeddings	0.0256	0.0003	0.0150	0.0150	0.0192	0.0233	0.000025	0.000125	0.000600	0.001925
7	BERT k=3 s_w	0.0221	0.0003	0.0150	0.0200	0.0167	0.0190	0.000075	0.000200	0.000500	0.001900

6 Discussions and Future Work

This report work poses the foundations for a new kind of retrieval (the proposed) model while gaining knowledge on many aspects of Neural networks, which continuously rise in popularity.

Specifically, issues regarding BERT and Word2Vec were studied and detailed, and a new Outlier function was developed starting from an existing one. The Cloud of Vectors model was theorized and multiple algorithms were proposed. Moreover, a preliminary implementation was realised to dig into the detailed model and its evaluation.

In short, the report represents the beginning of a complex, multi-faceted project that, will spawn further research on the matter.

The development of this work was not straightforward. For this reason, in section 1, a brief expository introduction on the importance of Information Retrieval in this day and age was presented, along with the main motivation for pursuing this work, as part of an experience in the field of Natural Language Processing and Language Theory. Section 2 mentioned all the influences that made this report possible along with the general tendency of the Information Retrieval field to pursue Passage Retrieval methods with Transformer Neural Networks. After that, the proposed model was detailed and its implementation was specified in chapters 3 and 4. The main goal of this report was to investigate if such a model made sense for different tasks and have a first approach at its empirical performances, shown in chapter 5.

Precisely in Section 5 we showed the results of all (the many) experiments that were run for the proposed model.

To summarize the observations conveyed, we can start from the pros: the Cloud Retrieval model surpasses the MoE baseline for every single metric in the LISA dataset and most of the metrics in the MS Marco dataset, in which we were expecting worse results because of the shorter queries. This fact by itself means that the model holds some merit and is semantically more meaningful than the baseline.

As for the Reranking tasks, the model stood its ground on both datasets as well: it is able to unilaterally improve all metrics on LISA even with respect to BM25's results, and improves most of the metrics, especially Recall, on MS Marco. These results suggest that the proposed Cloud Retrieval model is able to improve the recall at any step by retrieving those documents - that do not contain the exact terms that the user searched for- that are semantically close to the query.Therefore, a document that was part of a query's QRELS because of its semantic proximity, but not retrieved by BM25 because it didn't use the same tokens, will be dragged in the result set by the model's reranking score, hereby improving Recall.

In other words, a document is penalized by the tf-idf score (and therefore by BM25) if it does not contain words that are also present in the query, but still talks about a relevant topic or uses semantically close terms. The Cloud Retrieval model promotes this kind of outsider documents and reranks them to be a part of the result set, thus improving the recall metric.

However, it must be noted that the gain obtained by the model for the Ranked Retrieval task are still very small, and the improvement proposed on the bm25 results is also on the low side. The fact that the results are improved is nothing short of impressive, but we would like to see overall better performances. In order to improve said performances, we need to realize that the metrics gathered are nothing more than the result of a series of design decisions driven both by intuition and empirical results. During this report work we detailed a lot of critical points that still need to be study further in order to further evolve the cloud retrieval model, such as:

- Deeper study on the parameter k, possibly making it dynamic
- Applying the Query expansion algorithm 4
- Further testing of the document-wide algorithm 2
- Eventually developing a better Outlier metric

These items represent possible paths for future research on this model and possibilities to improve it.

Naturally, the scope of this report was to formalize and evaluate the first implementation of said model, and the goal was satisfied. However, finding a way to optimize the performances and overall reduce the query time would be crucial as well to massively bump up the number of experiments that can be carried on in a short amount of time. More testing, along with more studies on the effects of the parameters k and α, can be effective to understand their actual impact and converge to optimal values. Similar reasoning can be applied to the chosen aggregation functions. While these factors might not be as important as those mentioned before, they can still moderately improve performances and lead to better results overall, so it's important to keep them in check.

To conclude the model is very complex and deserves much more testing and parameter fine-tuning, not to mention the actual fine-tuning of the BERT Neural network since none was tested during this experimentation. We like to note that the Cloud Retrieval model improves results on both datasets posing the premise for a further future and deeper investigation.