Homework 1: Vector Search

Naomi Derel 325324994, Gili Cohen 326280815, Renana Shachak 213920010

02.07.2024

1 Faiss

With permission from the course staff, we kept our answers for the original question and not the updated requirements, as we finished it prior to the update.

1.1 Running Times

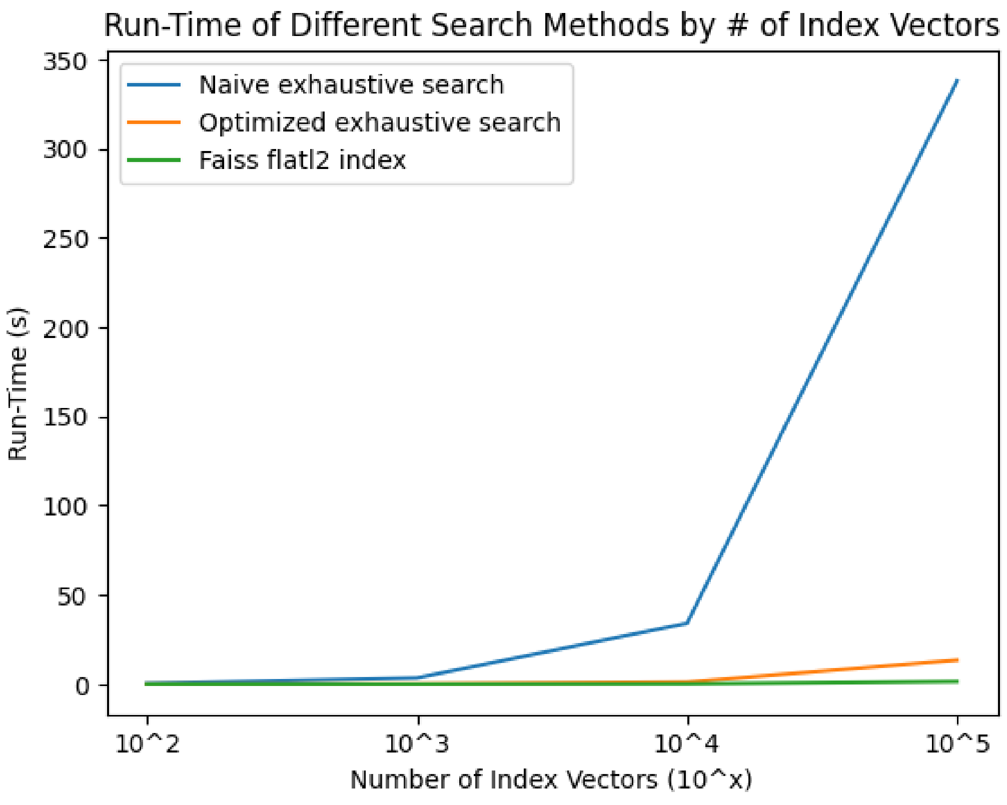

Figure 1.1.1: Running times by Number of Index Vectors

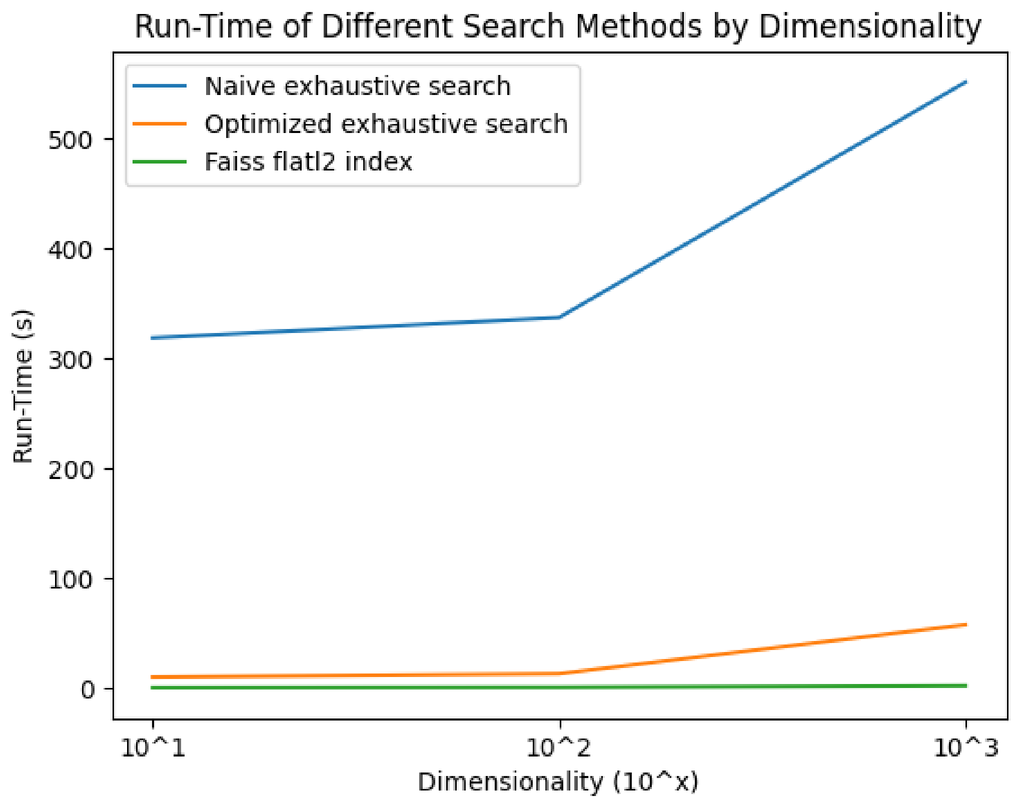

Figure 1.1.2: Running times by Dimentionality

1.2 Faiss LSH

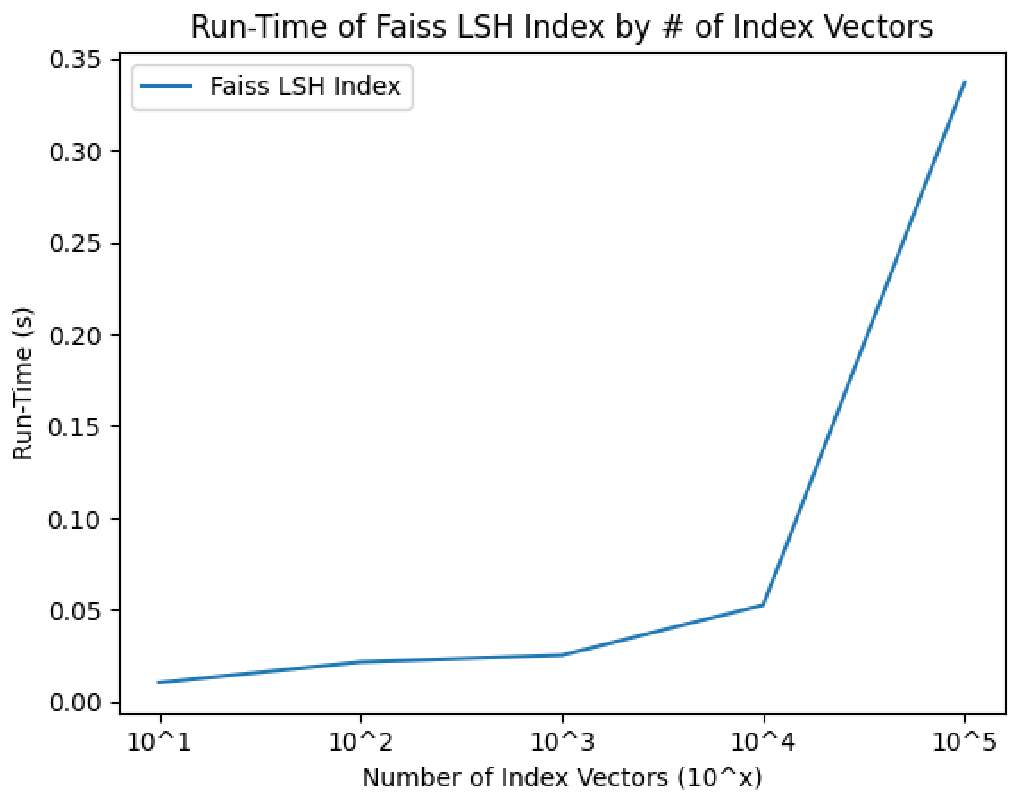

Figure 1.2.1: Running times by Number of Index Vectors

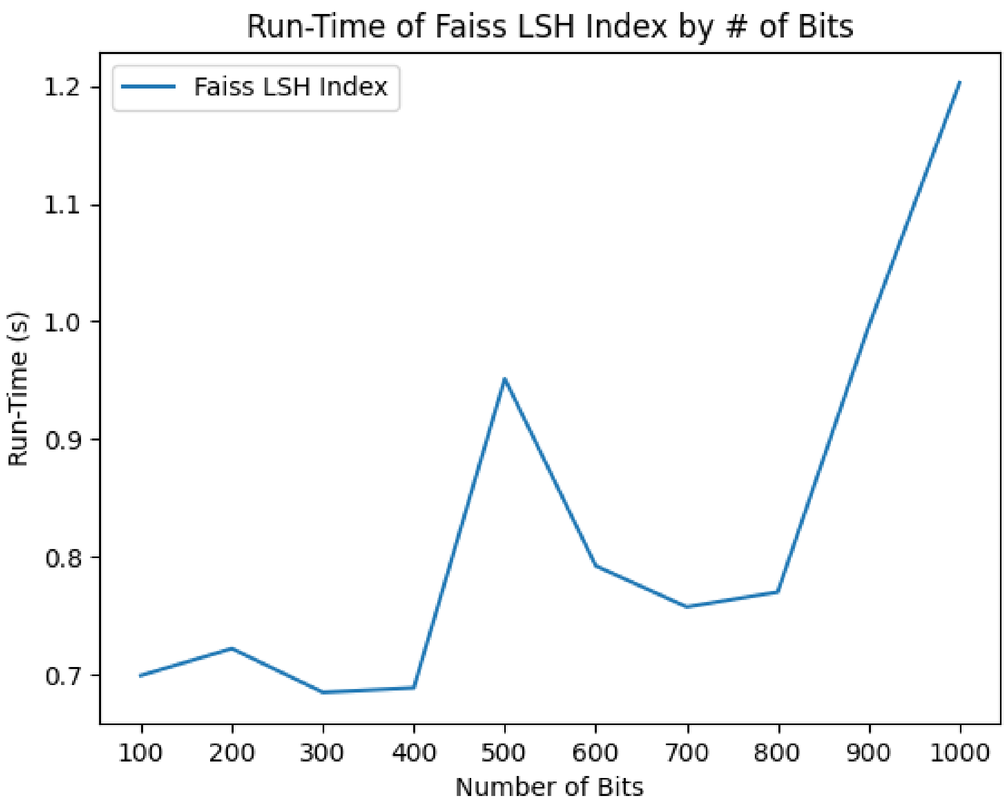

Figure 1.2.2: Running times by Number of Bits

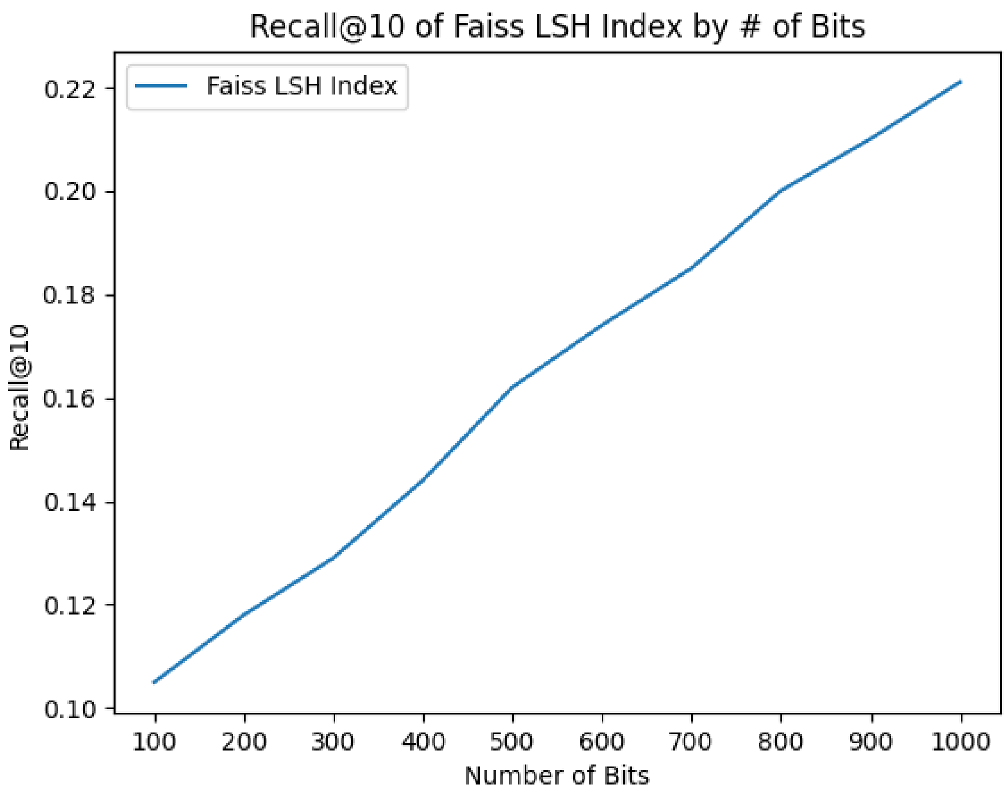

Figure 1.2.3: *Recall*@10 by Number of Bits

2 Implementing an Index

2.1 LSH vs. Native Exhaustive Search

1. Running time of the `semi optimized exhaustive search` function was: **26.8 seconds** wall time.
2. Running time of `build faiss lsh index` was: **5.54 seconds** wall time.
3. Running time of `faiss search` was: **0.187 seconds** wall time.
4. *Recall*@10 for the LSH index was: **0.138**.

2.2 Custom Indexing Algorithm

Implementation Description

For our solution, we decided to implement an index based on the IVF algorithm we saw in the lecture. Our intuition for this choice comes from the observation that the given datasets are all clusterized, which can be seen clearly when reducing the dimensionality of the data and plotting it:

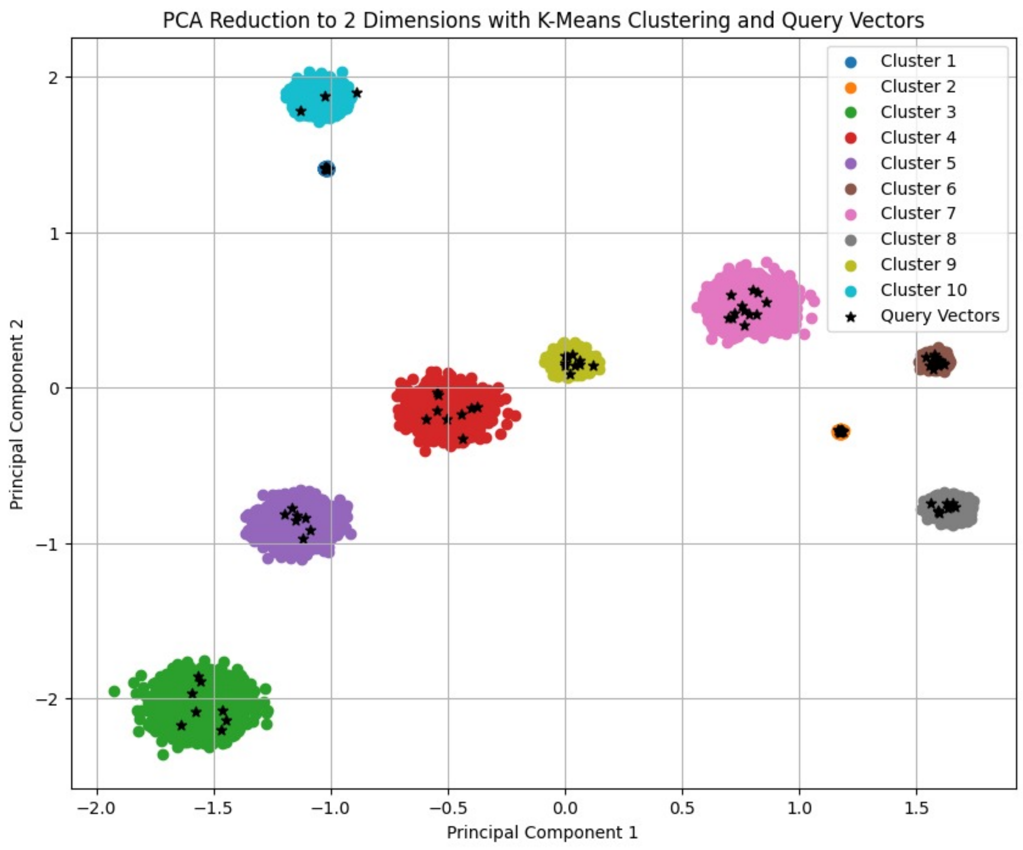

Figure 2.2.1: 2D PCA plot of the dataset clustered index vectors (& example query vectors)

The algorithm we implemented is as follows:

1. **Initialization:** We initialize the index with the minimal and maximal number of clusters to generate. We chose those to be 7 and 15 respectively based on the plots we generated, experimentation, and generalization to other datasets.

2. **Training:** We train the index by reducing the vectors to two dimensions with PCA, selecting the optimal number of clusters, and fitting a KMeans model to the data. The optimal number of clusters is chosen based on the elbow method on a SSE curve.

3. **Indexing:** We assign each index vector to the nearest centroid and store this assignment in a dictionary. Additionally, we store the centroids themselves.

4. **Searching:** Given a query vector, we reduce it to the 2D space generated by our PCA calculations, and find the nearest centroid to the query in said space. We then retrieve the original index vectors assigned to the nearest centroid (with their original dimensionality), perform an exhaustive search on these vectors and return the top k results.

Performance Measures

- **creating custom indexing algorithm:** 0.469 seconds wall time, less than half the time of section 2.1.1 (= 26.8 seconds)

- **searching with custom indexing algorithm:** 2.09 seconds wall time, less than a third of the time of section 2.1.1 (= 26.8 seconds)

- **recall@10 for custom indexing algorithm:** 1.0, more than 0.8.

3 Pinecone VectorDB and RAG

Github Repository: `https://github.com/NaomiDerel/Vector_Search_Lab`

3.1 Dataset Selection

We opted to use the Stanford Question Answering Dataset (SQuAD) as the dataset for this task. It consists of questions posed by crowdworkers on a set of Wikipedia articles, with corresponding reading passages. In total, SQuAD 1.1 contains 100,000+ question-answer pairs on 500+ articles.

The segments of contexts are repeated across questions, and after eliminating duplicates, we are left with 20,958 unique contexts. The segments are relatively short, with most segments containing less than 200 words, and the longest one being 635 words.

The answers for the questions are usually short, precise answers that can be found in the context, which makes them potentially easier to compare with the generated answer. However, there are also some questions that do not have an answer in the context.

The rationale behind choosing SQuAD is that it is a well-known dataset for QA tasks, but as we saw in a preliminary analysis, it contains a variety of ”trivia-like” questions of lesser known topics, which we quickly became convinced could mislead a regular QA model.

3.2 Anecdotal Model Evaluation

The standard QA model from cohere may fall short in answering factual questions in specific domains it might have not been exposed to enough. We present a few anecdotal examples of such cases:

1. **Question:** The College of Science began to offer civil engineering courses beginning at what time at Notre Dame?

   Correct Answer: the 1870s.

   (Shortened) Response without RAG: The College of Science at the University of Notre Dame began offering civil engineering courses in the fall of 2019. This expansion of the college’s curriculum was part of a broader effort to enhance the university’s engineering programs. . .

   Response with RAG: The College of Science at Notre Dame began offering early courses in civil and mechanical engineering in the 1870s.

   Clearly the response without RAG is very far off.

2. **Question:** How many departments are within the Stinson-Remick Hall of Engineering?

   Correct Answer: five.

   Response without RAG: There are six departments within the Stinson-Remick Hall of Engineering: Aerospace and Mechanical Engineering, Chemical and Biomolecular Engineering, Civil and Environmental Engineering and Earth Sciences, Computer Science and Engineering, Electrical Engineering, Materials Science and Engineering

   Response with RAG: There are five departments of study within the College of Engineering, which is housed in the Fitzpatrick, Cushing, and Stinson-Remick Halls of Engineering.

The response without RAG gave detail and seem to have invented a department. The response with RAG is more concise and accurate.

3. **Question:** How many student news papers are found at Notre Dame?

   Correct Answer: three.

   (Shortened) Response without RAG: There are two student newspapers at the University of Notre Dame: 1. The Observer: This is the main student newspaper on campus... 2. Irish Insider: This is a student-run sports newspaper that focuses specifically on Notre Dame athletics...

   Response with RAG: There are three student newspapers at the University of Notre Dame: The Observer, Scholastic, and The Juggler.

   The response without RAG is missing a newspaper, which might be due to a later addition of the paper or because of lacking information.

3.3 RAG Pipeline

- **Document reading and preprocessing:** We retained all the contexts from the SQuAD dataset, both in training and evaluation, into our corpus. We avoided duplicate snippets (most passages have multiple questions), and were left with 20,958 unique contexts. We embedded these documents using the `all-MiniLM-L6-v2` model, as shown in the tutorial.
- **Chunking documents for embedding generation:** As described in "dataset selection", the contexts for the questions are passages from Wikipedia articles, and so they are relatively short (we could think about this as if the passages are already pre-chuncked documents). Due to this, we believe further chunking might cause more loss of information than gain in noise reduction.
- **Embedding generation and insertion into Pinecone VectorDB:** We created a Pinecone VectorDB index using the API Key, and upserted the embeddings of the pure text of the documents into the index. In our dataset, we chose not to include additional metadata, as it was not informative further than the text itself.
- **Retrieval of relevant documents:** Given a specific query, we encode it and use the index to retrieve the top k most similar documents (we used $k = 3$). We combined the documents into a single source knowledge string.
- **Generating answers to given questions using the retrieved documents:** we augmented the query with the source knowledge and query strings in the following way:

  ```
  Using the contexts below, answer the query.
  Contexts: {source_knowledge}. If the answer is not included in the source
  knowledge - say that you don't know.
  Query: {query}
  ```

  We input the augmented query into the Cohere `command-r-plus` model to receive a response.

3.4 Insights

The examples shown present multiple potential types of error: blatantly incorrect information (ex. 1), hallucinated information (ex. 2) or lacking information (ex. 3). The common theme among these examples are domain specific, factual questions - which is what we aimed to solve with RAG.

We also conducted a small-scale evaluation of the RAG pipeline on a few (50) examples, and found that the RAG model, including the retrieval step, was faster than the standard QA model. We explain this by suggesting that the regular model without context might take longer to generate an answer it doesn't necessarily know, while the RAG model can answer very quickly once the context is provided.

Additionally, as supported by the examples we managed to fix, the average cosine similarity between the generated response and the answer was higher using RAG than the QA model. This strengthens our belief that the RAG model improves performance, although this is a flawed measure which might be effected by other factors (such as longer responses by the LLM which might have lower cosine similarity overall even if they contain the correct answer).

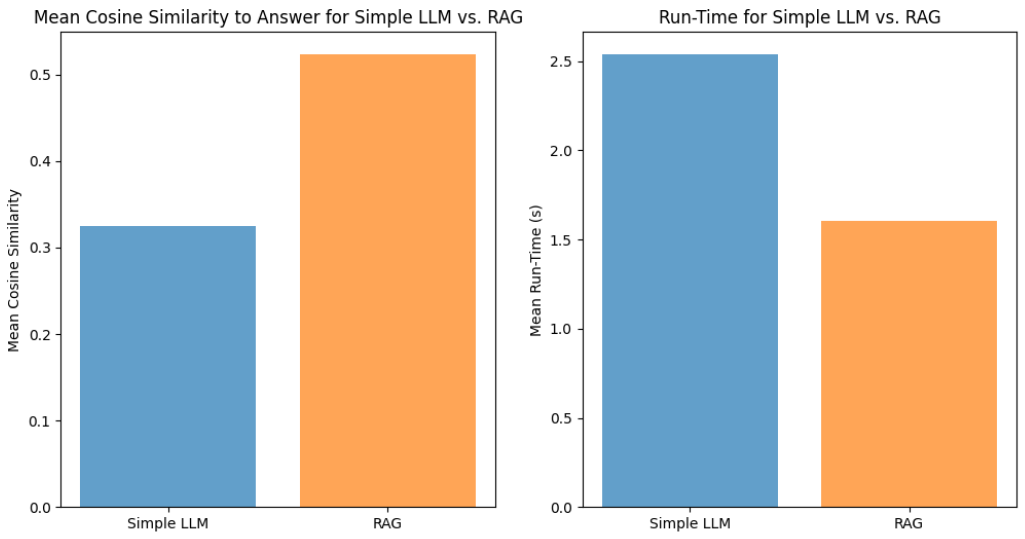

Figure 3.4.1: Comparison of RAG and QA model on 50 examples