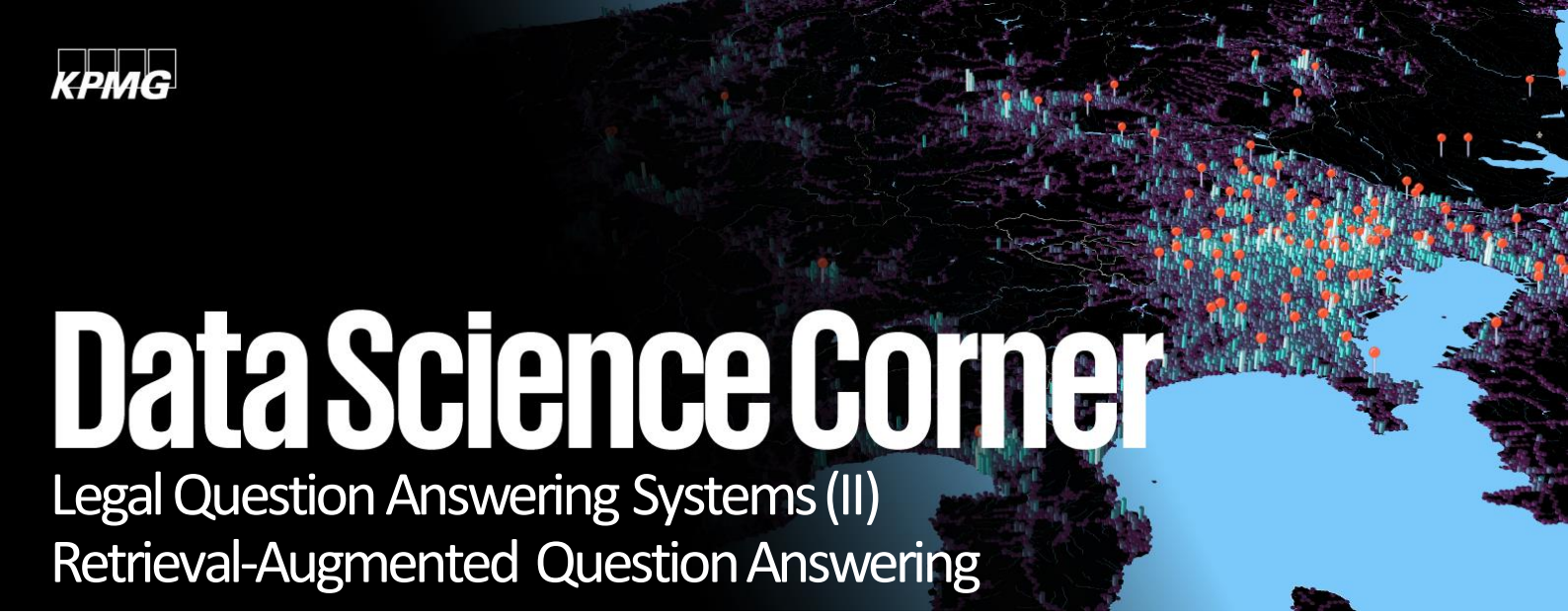

Data Science Corner

Legal Question Answering Systems (II)
Retrieval-Augmented Question Answering

As the legal industry continues to face increasing amounts of data, there is a growing need for efficient and accurate question answering systems that can navigate the complexities of legal language and provide insightful answers to legal queries. Retrieval-Augmented Question Answering (RAQA), a technique that combines information retrieval with generative language models, presents a promising solution to this challenge.

In this article, we introduce an RAQA system and report the results of our experiments involving different embedding methods for document encoding and a comparison of GPT-3.5 vs GPT-4 for generating answers to closed-form legal questions. Similar to the first article in this series, our experiments employ ContractNLI dataset[1] for reproducibility and accessibility.

Fundamentals of Information Retrieval

The main objective of an information retrieval (IR) system is to provide relevant information to users in response to their information need expressed as a textual query. Many theoretical models of how people search have been proposed to help design efficient IR systems. A classical example is the cognitive model proposed by Sutcliffe and Ennis[2], where the information seeking process is modeled as a cycle of the following.

1. Problem identification
2. Articulation of information needs
3. Query re-formulation
4. Results evaluation

Given a corpus comprised of multiple documents and a query, regarded as representative of the user's information need, a retrieval system generates a ranked list of corpus documents in descending order of relevance. Thus, the core problem of retrieval is how to estimate the degree of relevance, also known as relevance score, between a query and a document.

Many methods, also known as retrieval models, have been proposed to calculate the relevance score. While there is no single model that is considered fundamentally better than the others, three groups of models are well established: classical retrieval models, Learning to Rank (LTR) models, and neural retrieval models.

Classical Retrieval Models

Classical retrieval models, based on the probabilistic retrieval framework developed in the 1970s and 1980s, utilize exact keyword matching signals to design a relevance scoring function. In this group, BM25[3] is still regarded as a strong baseline. Classical retrieval models satisfy the following two properties:

1. The presence of a term in the document that matches a query term increases the score, and the score increases with the frequency of the term;

[1] Koreeda and Manning, 2021, "ContractNLI: A Dataset for Document-level Natural Language Inference for Contract"

[2] Sutcliffe and Ennis, 1998, "Towards a Cognitive Theory of Information Retrieval"

[3] Robertson et al., 1994, "Okapi at TREC-3"

2. Matching terms that are rarely present in the corpus, so called terms with a high inverse document frequency, increase the score to a greater degree.

Learning to Rank Models

Classical retrieval models usually do not require model training (though BM25 and other term weighting schemes may contain free parameters that can be tuned given training data). On the contrary, LTR models are models built with manually engineered features and trained with labelled relevance judgements.

Search systems that use one of these models can have hundreds of features. Many of these are based on statistical properties of terms and BM25 score is typically one of them. Other features are defined to capture user behavior (for example, how many times users issues a particular query or clicked on a particular document).

Depending on the type of training, an LTR model may output the degree of relevance of a document with respect to a query (pointwise approach), which document among two is more relevant (pairwise approach) or an entire ranked list (listwise approach). LTR models reached their zenith in the early 2010s before the deep learning revolution.

Neural Retrieval Models

In neural retrieval models, queries and documents are represented by high-dimensional numerical vectors, called embeddings. A geometrical proximity measure applied to the numerical representations of a query and a document, is used to calculate the "distance" between the document and the query and decide whether or not and how much the document is relevant to the query. These embeddings are generally learned by training a neural network with relevance judgements in an end-to-end manner.

In some cases, pretrained language models could also be used. However, it is important to remember that most inputs to search systems are composed of keywords rather than natural language sentences, which might deteriorate the reliability of the embeddings. Compared to classical retrieval models, neural search models are not limited by exact term matching. Because semantically related sentences are expected to have numerical vectors closer to each other, neural search models promise to find documents that are semantically related to the query, even if they do not contain the exact same keywords.

Compared to LTR models, terms of a query and a document are directly inputted in a neural network and no manual engineering of features is required.

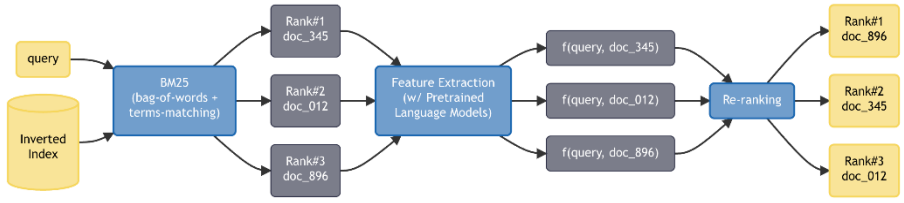

Figure 1: Neural Search Pipeline.

It is also common to combine different retrieval models. For example, documents may be ranked with a classical retrieval model, and top-k documents are passed to a second stage where they are re-ranked by a neural retrieval model. This is done because the application of neural retrieval models to a large set of documents may be computationally too expensive.

One typical example of this architecture is Azure Cognitive Search, which uses BM25 in the first stage, and when enabled, a proprietary neural retrieval model in the second stage.

Dataset

ContractNLI is a dataset for document-level natural language inference (NLI) on Non-Disclosure Agreements (NDA)s compiled by Stanford University. The dataset is compiled from the Internet and EDGAR, and aims to contribute to the process of AI-assisted contract reviews by providing more than 10k annotations on 607 contracts. Given a set of 17 hypotheses (full list given below) the annotations include which passages provide supporting or contradicting evidence for each hypothesis.

Problem Formulation

As explained in the first article of this Data Science Corner article series, we rephrased the original 17 hypotheses as questions and fed them to the RAQA system as user inputs, where the answer can take a value of "Yes", "No", or "Cannot tell". In other words, we created a closed-ended question answering task for each hypothesis.

ID	Original hypothesis	New question form
nda-1	All Confidential Information shall be expressly identified by the Disclosing Party.	Shall all Confidential Information be expressly identified by the Disclosing Party?
nda-2	Confidential Information shall only include technical information.	Shall Confidential Information only include technical information?
nda-3	Confidential Information may include verbally conveyed information.	Can Confidential Information include verbally conveyed information?
nda-4	Receiving Party shall not use any Confidential Information for any purpose other than the purposes stated in Agreement.	Can the Receiving Party use any Confidential Information for any purpose other than the purposes stated in Agreement?
nda-5	Receiving Party may share some Confidential Information with some of Receiving Party's employees.	Can the Receiving Party share some Confidential Information with some of Receiving Party's employees?
nda-7	Receiving Party may share some Confidential Information with some third-parties (including consultants, agents and professional advisors).	Can the Receiving Party share some Confidential Information with some third-parties (including consultants, agents and professional advisors)?
nda-8	Receiving Party shall notify Disclosing Party in case Receiving Party is required by law, regulation or judicial process to disclose any Confidential Information.	Shall the Receiving Party notify Disclosing Party in case Receiving Party is required by law, regulation or judicial process to disclose any Confidential Information?
nda-10	Receiving Party shall not disclose the fact that Agreement was agreed or negotiated.	Can the Receiving Party disclose the fact that Agreement was agreed or negotiated?
nda-11	Receiving Party shall not reverse engineer any objects which embody Disclosing Party's Confidential Information.	Can the Receiving Party reverse engineer any objects which embody Disclosing Party's Confidential Information?
nda-12	Receiving Party may independently develop information similar to Confidential Information.	Can the Receiving Party independently develop information similar to Confidential Information?
nda-13	Receiving Party may acquire information similar to Confidential Information from a third party.	Can the Receiving Party acquire information similar to Confidential Information from a third party?
nda-15	Agreement shall not grant Receiving Party any right to Confidential Information.	Does the Agreement grant Receiving Party any right to Confidential Information?
nda-16	Receiving Party shall destroy or return some Confidential Information upon the termination of Agreement.	Shall the Receiving Party destroy or return some Confidential Information upon the termination of Agreement?
nda-17	Receiving Party may create a copy of some Confidential Information in some circumstances.	Can Receiving Party create a copy of some Confidential Information in some circumstances?
nda-18	Receiving Party shall not solicit some of Disclosing Party's representatives.	Can the Receiving Party solicit some of Disclosing Party's representatives?
nda-19	Some obligations of Agreement may survive termination of Agreement.	May some obligations of Agreement survive termination of Agreement?
nda-20	Receiving Party may retain some Confidential Information even after the return or destruction of Confidential Information.	May the Receiving Party retain some Confidential Information even after the return or destruction of Confidential Information?

Table 1. ContractNLI dataset is composed of 17 different hypotheses for Natural Language Inference (NLI). We converted original hypotheses (statements) into question form and used them in evaluating different aspects of the described RAQA system

RAQA System Components

RAQA systems typically consist of two main components: an indexing pipeline and a question answering pipeline. Together, they enable the system to retrieve relevant information from a large collection of documents and generate precise answers to user questions.

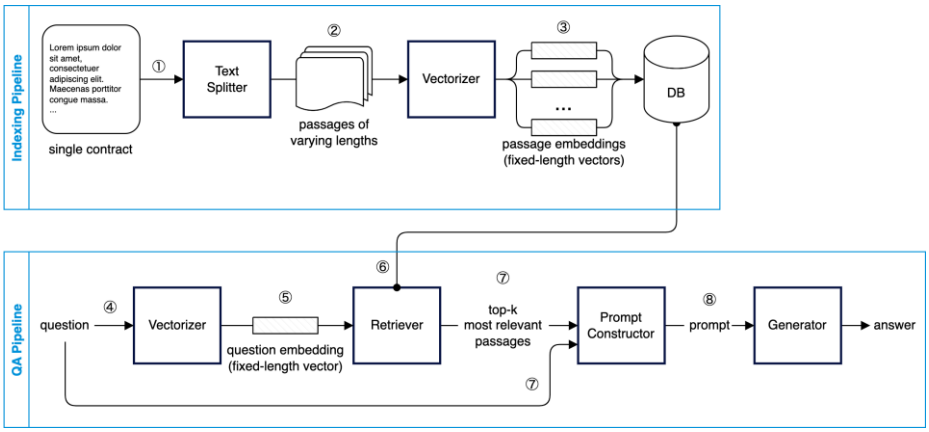

Figure 2. RAQA systems employ indexing and question-answering (QA) pipelines, where the indexing takes place offline and QA is executed online.

The indexing pipeline of RAQA systems focuses on transforming textual documents into dense vector representations. This process involves encoding each document into a high-dimensional vector space, where the proximity between vectors reflects the semantic similarity between the corresponding documents. Techniques such as sentence embeddings and transformer-based models are used to create these dense representations. This indexing approach enables efficient and accurate retrieval of relevant documents during the search process.

The question answering pipeline of RAQA systems, on the other hand, focuses on understanding and processing user queries to generate accurate answers. This pipeline begins by mapping the input query into a vector space that is identical to the one used for encoding documents. Its retriever unit then identifies and retrieves documents from the collection that are similar or relevant to the input query. The retrieved documents form the context for the Generator unit, which leverages powerful language models such as GPT-3.5 or GPT-4 to comprehend the meaning and intent behind the question.

The interaction between the indexing pipeline and the question answering pipeline is key to the success of a retrieval augmented question answering system. The indexing pipeline ensures that the documents are efficiently stored and retrievable, while the question answering pipeline uses encoded information to provide precise and contextually appropriate answers. The combination of these pipelines empowers the system to handle complex queries, process large volumes of data, and deliver accurate responses to user inquiries.

Next, let's take a closer look at the components of the indexing and question answering pipelines.

1. Text Splitter

Text-Splitter is responsible for breaking down input documents into meaningful passages. Since our main focus was on vectorizer, retriever, and generator, we chose to use the passages provided by the ConractNLI dataset without making any modifications. Here is how Koreeda and Manning describe this unit and how they obtained the passages for the ContractNLI dataset:

1. Extract plain text from the documents using pdf-struct[4]
2. Manually correct mistakes made by the tool
3. Use Stanza[5] to split each paragraph into sentences
4. Use regular expressions to further split each sentence at inline list items, e.g., at (a) or (iv)

[4] Koreeda, Yuta and Manning, 2021, "Capturing Logical Structure of Visually Structured Documents with Multimodal Transition Parser"

[5] Zhang et al 2021, "Biomedical and Clinical English Model Packages in the Stanza Python NLP Library"

2. Vectorizer

We experimented with seven different embedding methods for encoding documents and user questions. The following table summarizes key information about the chosen embedding methods.

While methods 1-5 produce dense vector representations, method-6 (BM25) is traditional keyword-based search, and method-7 (Azure Cognitive Search with Semantic feature) employs a hybrid approach that combines keyword-based search with semantic search.

#	Model	Context length	Embedding size	Model file size
1	OpenAI (text-embedding-ada-002)	8,191	1,536	N/A
2	Cohere (large)	512	4,096	N/A
3	STransformers (all-MiniLM-L12-v2)	256	384	134MB
4	STransformers (all-mpnet-base-v2)	384	768	438MB
5	STransformers (all-roberta-large-v1)	128	1024	1.42GB
6	Azure Cognitive Search (BM25)	N/A	N/A	N/A
7	Azure Cognitive Search (Semantic)	N/A	N/A	N/A

Table 2. List of vectorizer models used in the experiments

All of the three Sentence Transformers (Stransformers) are available on Hugging Face website.

- https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2
- https://huggingface.co/sentence-transformers/all-mpnet-base-v2
- https://huggingface.co/sentence-transformers/all-roberta-large-v1

Azure Cognitive Search (ACS) instances by default use traditional keyword-based search, where relevancy scores are calculated by the BM25 algorithm.

When their Semantic Feature is enabled, re-ranking is applied to the top-50 results, as scored by BM25.

3. Retriever

The role of the retriever is to identify and retrieve relevant documents or passages from a large collection of data based on the user's question. The retriever acts as the initial filter, narrowing down the search space to a subset of potentially relevant information.

Retriever uses various techniques and algorithms to match the query against the available documents and rank them based on their relevance. This could involve methods like keyword matching, as in BM25, semantic similarity, as in text-embedding-ada-002 followed by cosine or Support Vector Machines (SVM) calculations, or both as in ACS Semantic.

Retrieved documents serve as the context or input for the subsequent stages of the question answering pipeline, such as the generator. These stages utilize the retrieved information to generate or extract precise answers to the user's query. The retriever's role is critical in ensuring that the subsequent stages of the pipeline have access to the most relevant and useful information, which directly impacts the quality and accuracy of the system's answers.

Although computing cosine distances is the most common practice in measuring the semantic similarities between dense question and document vectors, Andrej Karpathy has recently advocated the use of SVMs (reference).

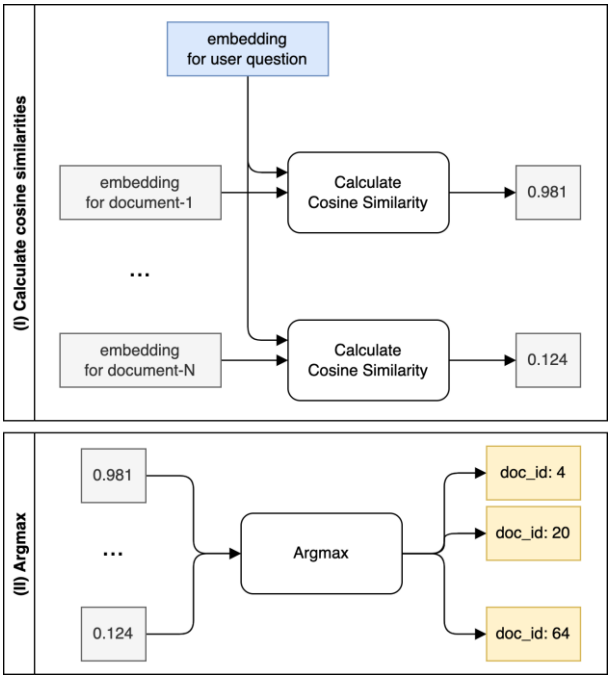

Figure 3. Cosine-based similarity ranking for the retriever

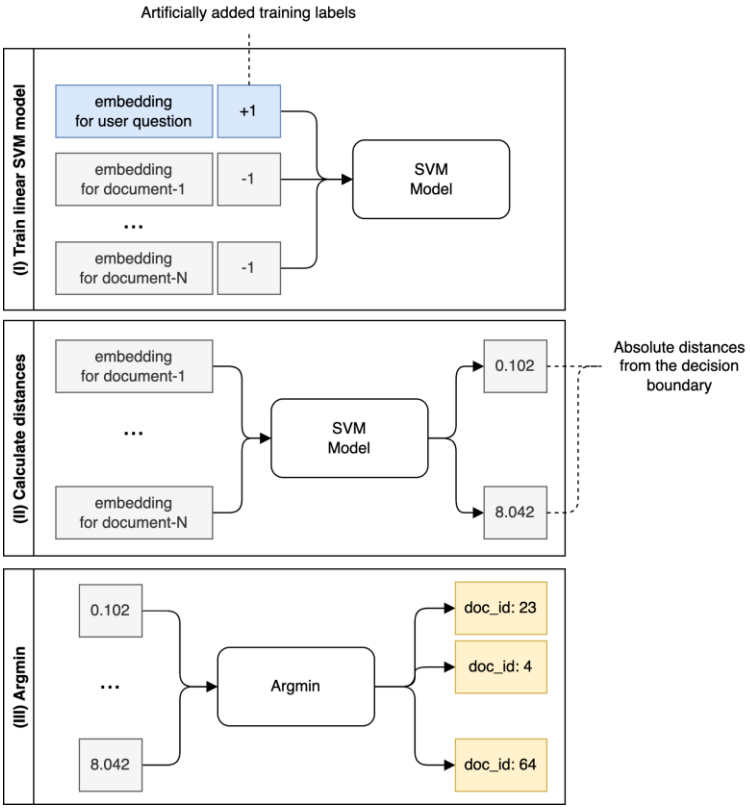

Figure 4. Support Vector Machines (SVM)-based similarity ranking for the retriever

3. Prompt Constructor

Given a set of seemingly relevant documents or passages, Prompt Constructor's role is to create the prompt, or the directive, for the subsequent generator. In our experiments, we used the following system and user prompts.

System Prompt

You answer questions about the terms of a particular contract.

- Your answers MUST BE `Yes`, `No`, or `I don't know`.
- Answer ONLY based on the provided terms given.
- If there isn't enough information in the provided terms, say `I don't know`.

User Prompt

- {Retrieved passage 1}
- ...
- {Retrieved passage k}

Question: {user question}

Answer:

5. Generator

We use GPT-3.5 and GPT-4 large language models offered in the Azure OpenAI Service. The job of the generator is to read provided context and generate a (textual) answer to the user question. The context includes user question and retrieved passages, i.e., seemingly relevant contract terms.

Experiments

We evaluated four major points in the design of RAQAs based on the recall@k metric.

1. Retrieval performance of different text embedding and vectorizer models
2. Retrieval performance of cosine-based vs SVM-based ranking
3. Reasoning capabilities of GPT-3.5 vs GPT-4
4. Impact of the number of retrieved documents on the performance of the Generator

Recall@k

The Recall@k metric measures the proportion of relevant results that are retrieved by the search system among the top k results returned.

For example, let's say that we have a search system that is given a query to retrieve relevant documents from a large database. We know that there are 100 relevant documents in the database. The search system retrieves the top 10 results for the query, and out of those 10, only 5 are relevant. In this case, the recall@10 score would be equal to 0.5 or 50%, because only 50% of the relevant documents are retrieved within the top 10 results.

Recall@k indicates how effectively the search system can retrieve relevant results. The higher the recall@k score is for a search system, the more effective it is at retrieving relevant results. Mathematically, this is given by:

$$Recall@k = \frac{num\_true\_positives@k}{num\_true\_positives@k + num\_false\_negatives@k}$$

1. Retrieval performance of different text embedding and vectorizer models

To measure the retrieval performance of different text embedding and vectorizer models, we focused on annotations with "Yes" or "No" cases and computed recall@k for k in {5, 10, 15, 20}. When compared to the other models in this setting, the Cohere (large) embeddings performed the best.

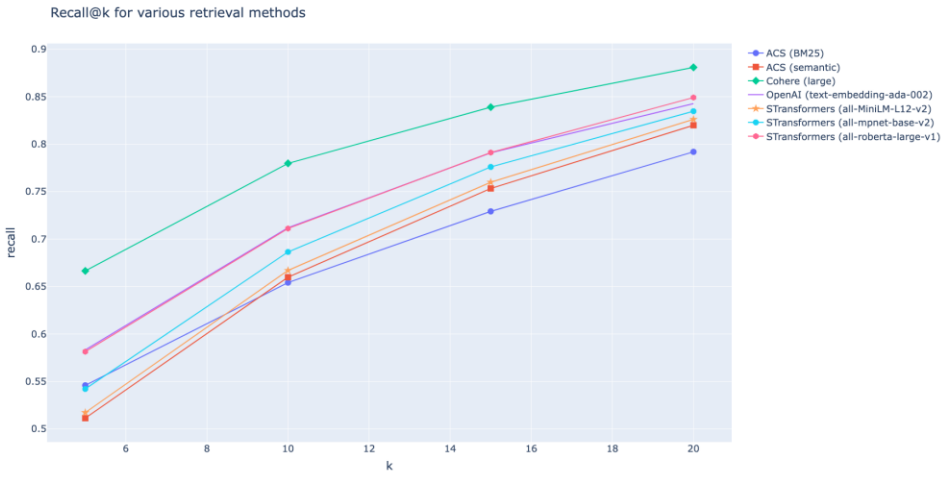

Figure 5. Recall@k for various retrieval models, vectorizers

Cohere (large) model outperforms other vectorizers including OpenAI's Ada text embeddings

2. Retrieval performance of cosine-based vs SVM-based ranking

We compared the retrieval performances of cosine and SVM-based similarities and could not observe any improvement with the use of SVMs. In fact, there was a significant drop in the performance of Cohere (large) embeddings when using SVMs.

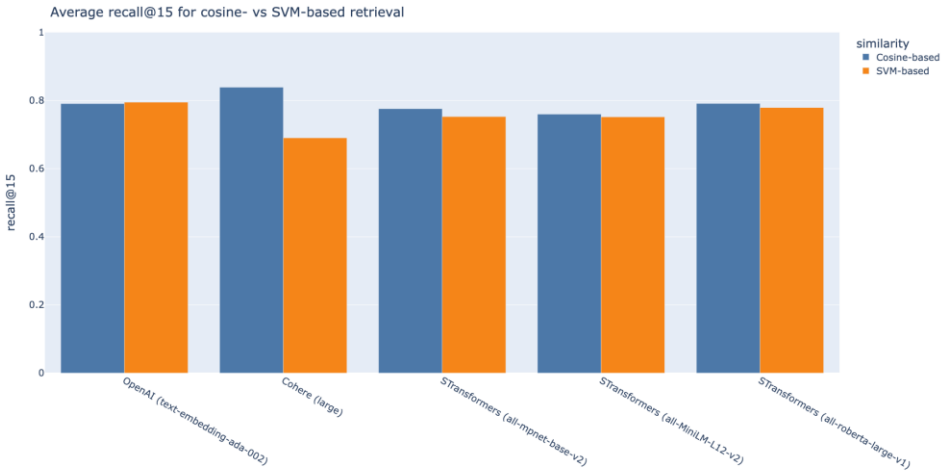

Figure 4. Average recall@15 for cosine- and SVM-based retrieval

SVM-based retrieval performs worse than cosine-based retrieval

3. Reasoning capabilities of GPT-3.5 vs GPT-4

To compare the reasoning capabilities of GPT-3.5 and GPT-4, we focused on annotations with "Yes", "No", "NotMentioned" cases and measured recall@15. We observed that GPT-4 performed significantly better than GPT-3.5.

We also observed that the performance of both Generators is the highest for the Entailment (answer is 'Yes') cases and the lowest for the Contradiction (answer is 'No') cases. In other words, the Generators are better at generating answers that are true than answers that are false based on the given context.

Careful readers might spot the fact that while all-roberta-large-v1 performed better than the other two Sentence Transformers in the previous experiment, it performed the worst among the three in this one. This is due to the differences in the focused annotations. In the previous experiment, we focused on annotations with "Yes" or "No" cases, while in this experiment, we focused on annotations with "Yes", "No", "NotMentioned" cases.

Rank	Generator	Retriever (cosine-based)	F1-score (No)	F1-score (NotMentioned)	F1-score (Yes)	Weighted F1-score
1	GPT-4	Cohere (large)	0.645	0.773	0.844	0.793
2	GPT-4	OpenAI (text-embedding-ada-002)	0.632	0.764	0.837	0.785
3	GPT-4	STransformers (all-MiniLM-L12-v2)	0.614	0.766	0.823	0.777
4	GPT-4	STransformers (all-mpnet-base-v2)	0.623	0.758	0.826	0.776
5	GPT-4	STransformers (all-roberta-large-v1)	0.625	0.752	0.819	0.771
6	GPT-4	ACS (BM25)	0.505	0.722	0.775	0.723
7	GPT-3.5	Cohere (large)	0.576	0.504	0.705	0.610
8	GPT-3.5	OpenAI (text-embedding-ada-002)	0.553	0.498	0.708	0.607
9	GPT-3.5	STransformers (all-MiniLM-L12-v2)	0.550	0.509	0.698	0.606
10	GPT-3.5	STransformers (all-mpnet-base-v2)	0.561	0.500	0.700	0.604
11	GPT-3.5	STransformers (all-roberta-large-v1)	0.562	0.495	0.701	0.603
12	GPT-3.5	ACS (BM25)	0.427	0.516	0.687	0.589

Table 3. End-to-end performances for different configurations

One factor contributing to the performance improvement is upgrading from GPT-3.5 to GPT-4.

4. Impact of the number of retrieved documents on Generator performance

To analyze the impact of the number of retrieved documents on the performance of the Generator, we used Cohere (large) embeddings, cosine-based ranking, and GPT-3.5 and computed weighted F1 scores. Results suggest that the more content we retrieve, the worse the Generator performs.

Rank	k	Weighted F1-score
1	2	0.650
2	8	0.641
3	15	0.610

Table 4. The effect of retrieving more documents for generator to process.

We observed a drop in the end-end performance (as measure by the weighted F1-score), as the retriver retrieved more documents.

Conclusion

In this article, we looked at a Retrieval-Augmented Question Answering (RAQA) system and provided a comparison of various retrieval models and text-to-text generators.

Key Takeaways

1. BM25 remains a valid baseline to start your journey in building an RAQA system, but it should not be your final destination.

2. The Sentence Transformers family consists of numerous members, one of which may be well-suited for your task. Among them, sentence-transformers/all-MiniLM-L12-v2 stands out as a strong baseline due to its compact size and powerful performance.

3. In our experiments, we noticed slightly lower retrieval performances when using SVM-based retrievals. However, we recommend considering SVM as an alternative to cosine-based retrieval and comparing both retrieval approaches.

4. If you are already using Azure Cognitive Search with its default BM25 ranking settings, consider enabling its Semantic Search feature and measuring the performance. You are likely to observe a slight improvement in retrieval performance.

5. Cohere can serve as a powerful and, at times, even superior alternative to OpenAI's text-embedding-ada-002.

6. The best end-to-end performance was achieved with the combination of Cohere embeddings and GPT-4.

7. The combination of BM25 and GPT-4 outperforms the combination of Cohere and GPT-3.5. This indicates that generators play a more significant role than retrievers in overall performance.

8. Generators are more accurate at generating answers that are true than answers that are false based on the given context.

9. When integrating retrievers with generators, system designers encounter an optimization challenge. On one hand, retrievers should retrieve a significant number of seemingly relevant documents to maximize the probability of finding the true relevant content. However, fetching excessive content can negatively impact generators' performance as they need to process and discern the true relevant information within the context. In our experiments, we achieved an improvement by reducing the number of retrievals from 15 to 2.

Future Work

Our future work will address the last point. The system that described in this article is essentially a unidirectional pipeline that starts with the ingestion of a question, continues with the retrieval of relevant passages by retriever, and ends with the generation of an answer by generator.

We would like to explore if better results can be achieved with a more iterative relationship between the Retriever and generator. By leveraging the superior cognitive abilities of recent Large Language Models, such as GPT-4, we plan to design generator as a search agent capable of autonomously choosing one action from a predefined list of possible actions. These actions may include the following.

1. **QUERY**: The generator queries the retriever with a set of keywords. This action may be useful in case Generator already has partial information for answering the question and believes that the remaining information can be obtained by a new query instead of reading further documents of the ranked list;

2. **NEXT**: The generator retrieves the next document from the ranked list. The idea is to limit the number of retrieved documents given to generator and let it decide autonomously, if further documents need to be retrieved in case the question cannot be answered.

3. **SUMMARIZE**: This is the last action where the generator is supposed to have collected complete information to answer the question.

We also would like to study which guardrails, such as answer plausibility validation, would be effective in reducing the risk of using these agents.

KPMG Ignition Tokyo, Inc.

Sercan Ahi

Senior Manager

Sercan.Ahi@jp.kpmg.com

Giacomo De Leva

Senior Manager

Giacomo.Deleva@jp.kpmg.com