Retrieval Augmented Generation: Leveraging Large Language Models and External Knowledge for Natural Language Search

RAVINDRA SADAPHULE, Johns Hopkins University, USA

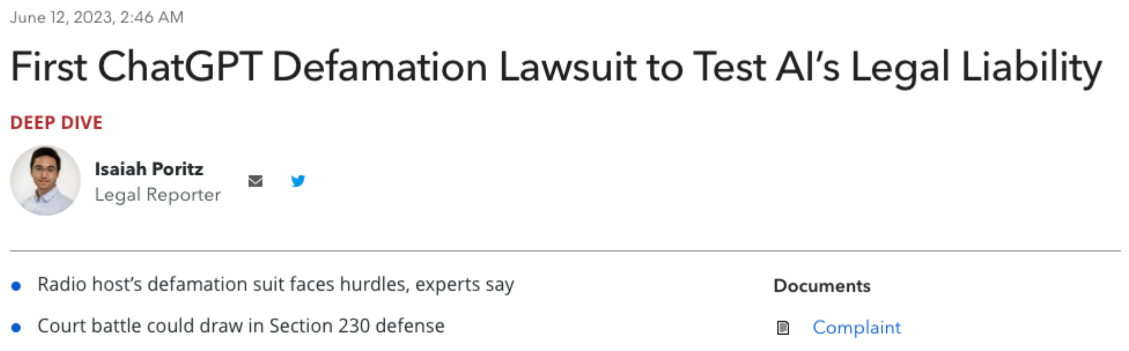
INTRODUCTION

Traditional information retrieval produces answers to the user's natural language query by searching through the corpus of documents or images crawled and indexed ahead of time. It uses technologies like inverted index, Tf-IDF, and okapi bm25 to search through millions of documents and produce relevant results in a sub-second. The results are produced as set of documents or images with highly relevant documents at the top. The user needs to read the title and teaser text to see what's relevant to their queries and click on relevant documents to get more information.

There are few known challenges that are inherent in traditional information retrieval systems. The first challenge is that the retrieval is done using the Bag of Words method. Given a set of n words in the query, the system looks for all permutations and combinations of the words in the documents and surfaces the results. For example, the system cannot differentiate between queries like "chocolate milk" and "milk chocolate." The second challenge is that the answer that the user is looking for could be scattered through multiple documents. This requires the user to make additional effort to comb through the documents, and aggregate the response from various documents in the search result to generate the result.

With the recent advances in large language models, we could address the challenges described above by combining large language models and information retrieval engines to generate a unique answer that is enough for the user to satisfy his/her intent. However, if we just use large language models out of the box, it won't yield relevant and up-to-date results. Traditional seq2seq models often generate generic, templated responses due to insufficient external world knowledge and reasoning capabilities. These models have been trained on the public corpus. They are not aware of unique enterprise knowledge. As a result, Large language models suffer from hallucinations, and their knowledge base has a cut-off date (Chat GPT has a cut-off date of Sep 2021). As a result, they can not be used to search for real-time information that enterprises typically have.

The consequences of hallucinations have been so severe that few companies have been sued. Here are a few examples.



June 12, 2023, 2:46 AM

First ChatGPT Defamation Lawsuit to Test AI's Legal Liability

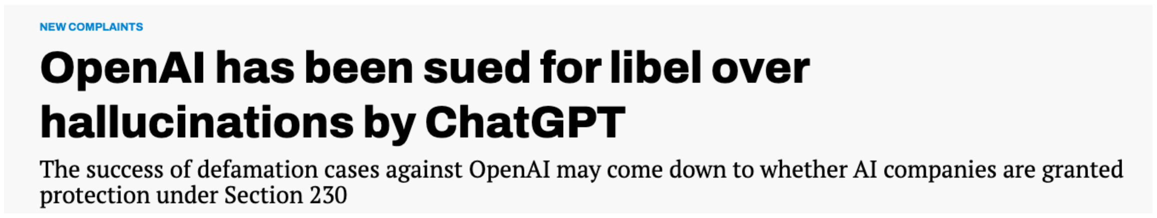
DEEP DIVE

Isaiah Poritz
Legal Reporter

- Radio host's defamation suit faces hurdles, experts say
- Court battle could draw in Section 230 defense

Documents

Complaint



NEW COMPLAINTS

OpenAI has been sued for libel over hallucinations by ChatGPT

The success of defamation cases against OpenAI may come down to whether AI companies are granted protection under Section 230

The primary reason behind these hallucinations is that LLMs are not grounded. Traditional large language models like GPT-4 embed general world knowledge base and natural language understanding together as a single module. The training of these models is very expensive in terms of time and cost. GPT-3 model training on A100 GPU costs about $100M and one month. These large language models often have a cut-off date. (e.g., GPT-3 has a cut-off date of Sep 21, 2022). Hence they are unable to comprehend the latest changes in world events.

In this paper, we show that by combining information retrieval and large language models. We can address the limitations of large language models and traditional information retrieval systems and generate a single answer that's highly relevant to user's query. This is still a very nascent technology known as **Retrieval Augmented Generation (RAG).** RAG models first retrieve relevant context documents from an external knowledge source like Wikipedia or enterprise corpus in response to the input text. These retrieved documents provide key facts, concepts, entities, and other knowledge that can inform and ground the text generation process. The retrieved documents are encoded and provided as additional input to the seq2seq model, which then uses them to generate an output response.

The proposed research will evaluate different encoder architectures and retrieval mechanisms to identify optimal techniques for retrieving salient, relevant external knowledge and integrating it into the text-generation process. The goal is to develop RAG models capable of producing more factual, knowledgeable, and coherent natural language compared to traditional seq2seq models. We'll compare results with and without RAG and show the unique strengths of the RAG approach.

Additional Key Words and Phrases: Large Language Models, Retrieval Augmented Generation, Semantic Search, Pretrained Models

Goals:

Design an informational retrieval system that overcomes the limitations of okapi bm25 based system by

- Using Embedding Based similarity instead of BOW (bag of Words) model
- Summarize the answers by generating one answer from the documents matching user’s query.

RESEARCH

Retrieval Augmented Generation is a promising technique to overcome traditional BOW word challenges and lack of summarization associated with traditional OKAPI BM25-based Information retrieval Systems. Here a pre-indexed knowledge base is used to enhance the capabilities of a neural text generator. The key idea is to retrieve relevant context from the knowledge base to condition the language model, guiding it to generate high-quality, knowledgeable responses. Some novel methods include dense vector indexing approaches like FAISS that allow efficient retrieval from huge corpora and dense representations like BERT embeddings that capture semantic similarity for retrieval. The indexer and retriever modules can be trained end-to-end with the generator using methods like weakly supervised training or reinforcement learning.

A key focus in recent work is developing more powerful retriever modules using dense representations and advanced training techniques. For example, Gao et al. (2021) pre-train a dual-encoder retriever with a contrastive loss to maximize the embedding similarity between context and response pairs. They show significant gains over TF-IDF retrieval, reducing the gap with human performance by 55-72% across several dialog tasks. Their RETRO model achieves state-of-the-art results on the Wizard of Wikipedia benchmark.

For example, RAG (Lewis et al. 2020) [1] trains a dense retriever on pairs of contexts and responses using binary classification. The retrieved contexts are concatenated with the input to condition a T5 language model. This improves the factual consistency and logical coherence in the generated text.

Another thrust is enhancing the generator model by conditioning it on retrieved knowledge. Fabbri et al. (2021) [3] show that concatenating contexts can improve consistency improve, consistency but hurts fluency and coherence. They introduce a multi-encoder model where a long-term encoder separately summarizes dialog history and a short-term encoder attends to recent context. This improves engagement and information flow compared to single-encoder approaches in conversational tasks.

In terms of training, Ouyang et al. (2022) [4] find that interleaved training of the retriever and generator leads to improved performance over staged training. but find end-to-end fine-tuning is unstable. A curriculum learning approach is proposed to address this, where simple retrieves are first learned before increasing difficulty.

Lee et al. (2021) [5] augment a summarization model with retrieved evidence and train the system end-to-end using human ratings. They can generate high-quality summaries that are both concise and grounded in source facts. The approach reduces hallucination and improves faithfulness compared to ungrounded models.

There is also work on more specialized retrieval augmentations like fact-checking (Lee et al. 2021) [5], numerical reasoning (Zhang et al. 2021)[13] , and table-text generation (Chen et al. 2020)[7].

Overall, this is a rapidly evolving sub-field, and many open problems remain around scalability, stability, and versatility.

HYPOTHESIS

We believe that RAG-based information retrieval outperforms the traditional okapi bm25 approach for QnA style queries because of the fundamental difference in how the system computes relevance score.

Traditional information retrieval systems use okapi bm25 algorithms to rank the documents for a given query

$$score(D) = \sum_{i=1}^{n} IDF\left(q_i\right) \times \frac{f\left(q_i, D\right)\times\left(k_1+1\right)}{f\left(q_i, D\right)+k_1\times\left(1-b+b\times\frac{|D|}{avgdl}\right)}$$

Where

- score(D) is the score of document D
- qi is the ith query term
- f(qi,D) is the frequency of term qi in document D
- |D| is the length of document D
- avgdl is the average document length in the collection
- k1 and b are free parameters, usually chosen, for example, as 1=2.0, k1=2.0 and k1=0.75, b=0.75

The algorithm first scans word tokens in query and then for each word, it computes term frequency(TF) of each word in each document and inverse document frequency(IDF) of each word across document corpus and computes score them in such a fashion so that retrieval favors rare words more as compares to common words.

Okapi BM25 Algorithm:

Input:

- Document (D)
- Query (Q) consisting of terms $(q_1, q_2, ..., q_n)$
- Collection of documents (C)
- Parameters(k_1) and (b)

Output:

- Relevance score of document (D) for query (Q)

Steps:

1. **Initialization:**
    - Set $(score(D) = 0)$
    - Calculate $(avgdl)$, the average document length of all documents in (C)

2. **For each term** (q_i) **in** (Q):
   a. Calculate term frequency$(f(q_i, D))$ of (q_i) in (D)
   b. Calculate the document frequency$(df(q_i))$ of (q_i) in (C)
   c. Calculate the inverse document frequency$(IDF(q_i))$using:

   $[IDF(q_i) = \log log\left(\frac{N-df(q_i)+0.5}{df(q_i)+0.5}\right)]$

   where (N) is the total number of documents in (C)

   d. If $(IDF(q_i) < 0)$, set $(IDF(q_i) = 0)$ (to ensure non-negative scores)
   e. Calculate the term score for (q_i) in (D) using:

$$[term\_score(q_i, D) = IDF(q_i) \times \frac{f(q_i,D)\times(k_1+1)}{f(q_i,D)+k_1\times\left(1-b+b\times\frac{|D|}{avgdl}\right)}]$$

   f. Update the document score:
   $[score(D) = score(D) + term\_score(q_i, D)]$

3. **Return** $(score(D))$

While okapi bm25 works very well for short queries, it lacks a semantic understanding of long queries. BM25 is a bag-of-words model, meaning it doesn't understand the semantic relationships between terms. For long queries, especially those that are complex or phrased as natural language questions, the lack of semantic understanding can lead to a suboptimal ranking of documents.

Ideally, we would like to understand the meaning behind the entire search query and search document corpus with semantic meaning. This is where a semantic phrase or sentence embedding comes in handy.
Sentence embeddings are fine-tuned on top of word embeddings. While word embeddings represent individual words in a high-dimensional space, Sentence embeddings represent entire sentences or even paragraphs as vectors in a similar high-dimensional space. Sentence Embeddings Can be generated using various methods:

- Averaging Word Embeddings: A simple method where the word vectors for all words in a sentence are averaged.
- TF-IDF Weighted Averaging: Words are weighted based on their TF-IDF scores before averaging.
- RNNs and LSTMs: Sequence models that consider the order of words.

- Transformer-based Models: Models like BERT, GPT, and Roberta generate contextual embeddings, where the representation of a word is based on its surrounding words in a sentence.
- Doc2Vec: An extension of Word2Vec that learns to represent entire sentences or documents.

Here is how sentence-based semantic similarity computes the similarity score between query and document

$$cosine\_similarity(A, B) = \frac{A \cdot B}{||A||_2 \times ||B||_2}$$

Where

- $A \cdot B$ is the dot product of the vectors A and B.
- $||A||_2$ is the L2 norm (Euclidean norm) of vector A, and $||B||_2$ is the L2 norm of vector B.

This equation gives a value between -1 and 1, indicating the cosine of the angle between the two vectors. A value of 1 means the vectors are identical, 0 means they are orthogonal (no similarity), and -1 means they are diametrically opposed (though in the context of text embeddings, we'll rarely see a value of -1).

Embedding-Based Semantic Search Algorithm

Input:

- Query (Q)
- Collection of documents ($D = \{d_1, d_2, ..., d_n\}$)
- Pre-trained embedding model (M). (BERT, XML-Roberta, etc)

Output:

- Ranked list of documents based on their semantic similarity to the query

Steps:

1. Embedding Generation:

a. Use the sentence embedding model (M) to generate an embedding vector(q) for the query (Q).
b. For each document (d_i) in (D), use (M) to generate an embedding vector(d_i).

2. Compute Similarities:

For each document (d_i) in (D):

a. Compute the cosine similarity between(q) and (d_i) using:

$$similarity(i) = \frac{q \cdot d_i}{||q||_2 \times \left\|d_i\right\|_2}$$

b. Store the similarity score $(similarity(i))$ for document (d_i).

3. Ranking:

Sort the documents in (D) based on their similarity scores in descending order.

4. **Return Results:**

Return the ranked list of documents sorted by similarity score

This algorithm provides a high-level overview of how embedding-based semantic search works. In practice, there are optimizations and additional considerations, such as:

Using approximate nearest neighbor (ANN) search for scalability when dealing with a large number of documents.

Incorporating context-aware embeddings for better handling of polysemy.

Combining embedding-based scores with other relevance signals for a more comprehensive ranking.

METHOD:

Our approach involves first computing sentence embedding for each document and storing it in the index. Then when performing a search query, we first compute sentence embeddings for the query and compute cosine similarity between the query embedding vector and document embedding vector, and order the results by similarity score.

Here is an example:
Imagine a user is searching for articles on "The health benefits of regular aerobic exercise."

Traditional Keyword-Based Search with BM25:
Query Tokenization: The search engine breaks down the query into individual terms: "health," "benefits," "regular," "aerobic," and "exercise."
Document Scoring: BM25 scores documents based on the frequency of these terms, their inverse document frequency, and other factors.
Limitations:
Synonyms: An article titled "Advantages of consistent cardiovascular workouts" might be highly relevant but could be ranked lower because it doesn't use the exact terms from the query.
Lack of Semantic Understanding: BM25 doesn't understand that "advantages" can be synonymous with "benefits" or that "cardiovascular workouts" are a form of "aerobic exercise."
Long Queries: If the query is lengthy or phrased in a complex manner, BM25 might not give adequate weight to the most crucial terms, leading to less relevant results.

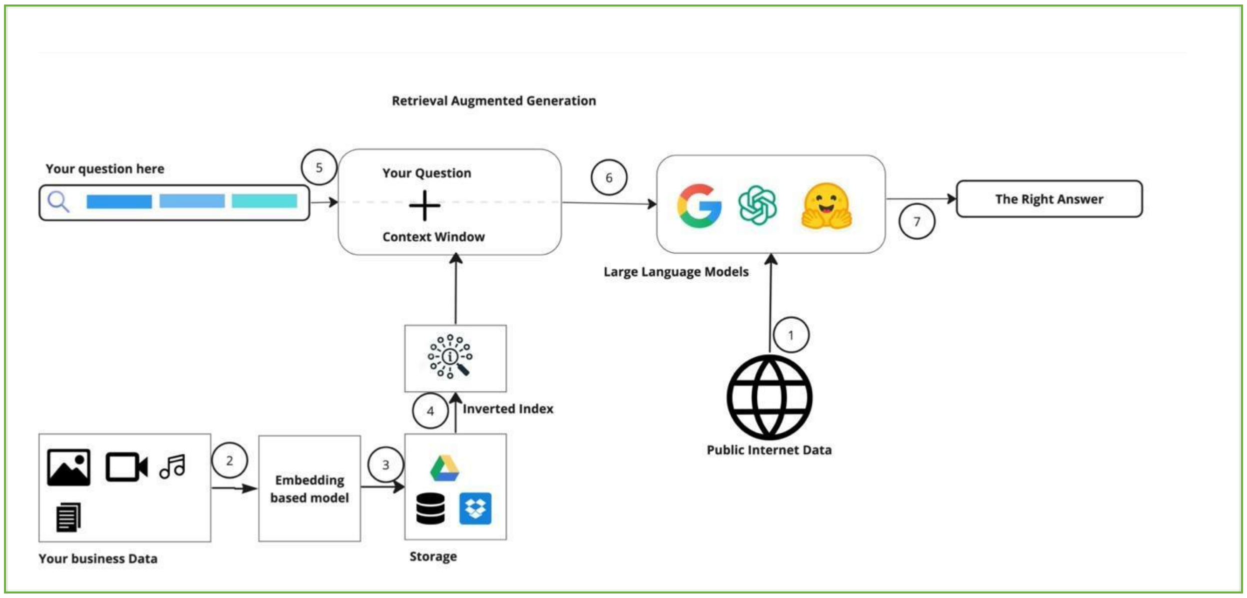
Embedding-Based Search:
Vector Representation: Both the query and the documents are transformed into vectors in a high-dimensional space using word or sentence embeddings.
Semantic Similarity: The search engine computes the cosine similarity (or another distance metric) between the query vector and document vectors.
Advantages:
Understanding Synonyms: The embeddings can capture the semantic similarity between "benefits" and "advantages" or between "aerobic exercise" and "cardiovascular workouts."

Now that we have solved the semantic search problem, lets us turn our attention to summarizing the answer. We would like to leverage Large Language models for summarization. Language model-based methods, especially large language models (LLMs) like GPT-3, BERT, and T5, have shown significant advancements in various natural language processing tasks, including summarization. LLMs are trained on vast amounts of text, enabling them to capture and understand context better than traditional methods. This understanding allows them to generate summaries that retain the essential information and context from the retrieved search results. Models like T5 are designed as sequence-to-sequence models, which means they can take a sequence of text (e.g., search results) as input and produce another sequence (e.g., a summary) as output. This architecture is inherently suited for summarization tasks. LLMs can be fine-tuned on specific summarization tasks after being pre-trained on large corpora. This two-step training process allows them to leverage both general language understanding and task-specific knowledge, leading to high-quality summaries.

Here is how we'll use LLM for summarization

- When the user types in a query, first, we'll generate an embedding for the query and search the inverted index using the embedding search method.
- We'll get semantically related documented as search results which are much more relevant than traditional bm25 results. We'll retrieve textual metadata from these embeddings.
- We'll then collate metadata from the first 10 documents of the search result and create a context for the LLM prompt. We'll add the question as a prompt and search results as a context window and then issue the query to a large language model. This context window helps ground the LLM. We'll ask LLM to generate the final answer only in the context of the current window. LLMs will help us generate one answer in a way that's more intuitive and appealing to the user. LLMs can also augment the data from the public internet but in the context of search results that we retrieved.

Here is a block diagram that depicts workflow



1. Select a pre-trained LLM that we would like to use. I have used a pretrained Open AI GPT-3 generative text model for this exercise. This model used Transformer architecture and has 175B parameters. This model has been trained mixture of licensed data, data created by human trainers, and publicly available data. A typical training of GPT-3 models takes about a month on an A100 GPU, and it costs about $100M dollars
2. Select data that the user needs to search on. We have selected s Wikipedia corpus.
3. Use a sentence embedding model to create sentence embedding for each model and store it inverted index (Elastic Search v8.8).
4. We'll write a script and generate OpenAI embeddings for each Wikipedia document and store embeddings in Elastic Search
5. At query time, when the user types in a query, we'll create open embeddings and search the elastic search corpus using the ANN (Approximate Nearest Neighbor) query to fetch the text content of relevant documents.
6. We'll create an open prompt with the user's query as prompt, collate retrieved documents as prompt context, and "Generate a summary" as a command and send it to open AI API/
7. OpenAI API generates a summary of all documents and returns it back, which we then present to the user.

How we'll measure success:

Quantitative Measurements:

We'll use industry-standard metrics like Precision@K, NDCG, and Human judgments to evaluate the performance of RAG based approach against OKAPI bm25 and prove that RAG performs better

Precision@K: Precision@k measures the proportion of relevant items among the top-k items in the ranked list of results. A value of 1 indicates that all top-k results are relevant. A value of 0 indicates that none of the top-k results are relevant. It's particularly useful when the user is only interested in the top-k results, such as the first page of search results or the top-k recommended items.

$$Precision@k = \frac{Number\ of\ relevant\ items\ in\ the\ top-k\ results}{k}$$

NDCG@K (Normalized Discounted Cumulative Gain):

NDCG is a metric that evaluates the quality of a ranked list of results, considering the relevance of each result and penalizing relevant results that appear lower in the list.

DCG (Discounted Cumulative Gain) :

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{(i+1)} \qquad \text{eq. 1}$$

IDCG (Ideal Discounted Cumulative Gain):

This is the DCG@k value for the ideal ranking. It's computed similarly to DCG but with results sorted by their true relevance.

NDCG:

$$NDCG@k = \frac{DCG@k}{IDCG@k} \qquad \text{eq. 2}$$

Where:

- (rel_i) is the relevance score of the result at position (i).

Qualitative Measurements:

We'll perform human judgments either by using a crowdsourced platform like APPEN or Mechanical Turk or in a group of small people. We'll show search responses from 2 candidates (RAG and OKAPI BM25) to the user for around 100 queries, and for each query, we will ask a user to rate if RAG is better or OKAPI BM25 is better. We'll aggregate the results and present conclusions.

APPLICATION

- I’ll be using OpenAI’s API to query GPT -3.5 LLM embeddings API for inference. I have OpenAI subscriptions. I have checked rate limiting by issuing calls using Open AI tokens from the Python notebook. I can get up to 100 requests per minute and 50,000 tokens per minute. This is good enough for my experiment
- I’ll be using Elastic Search open source version 8.10 as an inverted index. I’ll be downloading it locally on my machine and creating a single node single shard index
- I’ll be indexing 1,000- 10,000 documents from specific public domains (Wikipedia) and Indexing them in elastic search
- I’ll create a Python Streamlit application to showcase how Retrieval Augmented Search works end to end.

Data Source:

Wikipedia: https://cdn.openai.com/API/examples/data/vector_database_wikipedia_articles_embedded.zip

Number of Documents: 35000

Fields:

Column Name	Data Type	Description
Title	String	Title of the page
Description	String	Content of the page
URL	String	URL of the page
Title_vector	Dense Vector	Title Embeddings
Contenet_Vector	Dense Vector	Content Embeedings

Link to Python Notebook: https://github.com/rsadaphule/jhu-aaml/blob/main/research/RAG_Wikipedia.ipynb

Note: Please note that the code uses trail version of Elastic Search which is valid for 14 days. After 14 days you must renew the trial or sign up for a trial with another email id.

I have also recorded a demo Video at: https://youtu.be/V5LSGS87oZE

Set up instructions:

1. Download Anaconda from https://www.anaconda.com/download
2. Create a virtual environment and choose Python 3.8 version
3. Download the Python notebook from GitHub https://github.com/rsadaphule/jhu-aaml/blob/main/research/RAG_Wikipedia.ipynb and put it in the local drive
4. Install the Jupyter Notebook app and open the above Notebook
5. Follow all the instructions within the notebook to install Elastic Cloud, download the dataset, index the dataset, and create streamlit app
6. Run the streamlit all
7. Collect a set of 100 queries for which you run the evaluations.
8. Crowdsource the evaluations and collect them in an Excel document.
    a. Eval 1: Ask users to judge the relevance score for a combination of each document and head, torso and tail queries on a 1-4 scale and report it
    b. EVal 2: Ask users to judge if results on the left side (bm25) or results on the right side look better on 1-5 scale. Ask them to report aesthetic score in excel
9. Compute mean NDCG, mean precision@1, and mean aesthetic score and report it for short and long queries

Experiment Results

HYPOTHSIS FORMULATION

NDCG mean of the current BM25 search system is less than 0.6. I believe that the NDCG mean of the RAG system using exact same corpus would be greater than 0.6. Using 1000 query samples, I measured that the NDCG mean is 0.8 with a standard deviation of 2.5. I would like to state with 95% confidance that the RAG system's NDCG mean is better than BM25 based system.

Here is the hypothesis formulation

Null Hypotheis: $\mu 0 < 0.6$

Alternative Hypotheis: $\mu 0 > 0.6$

This is once sided test.

n=100, X-bar = 0.7, sd = 2.5, alpha = 1-0.95 = 0.05

Lets compute Z-score

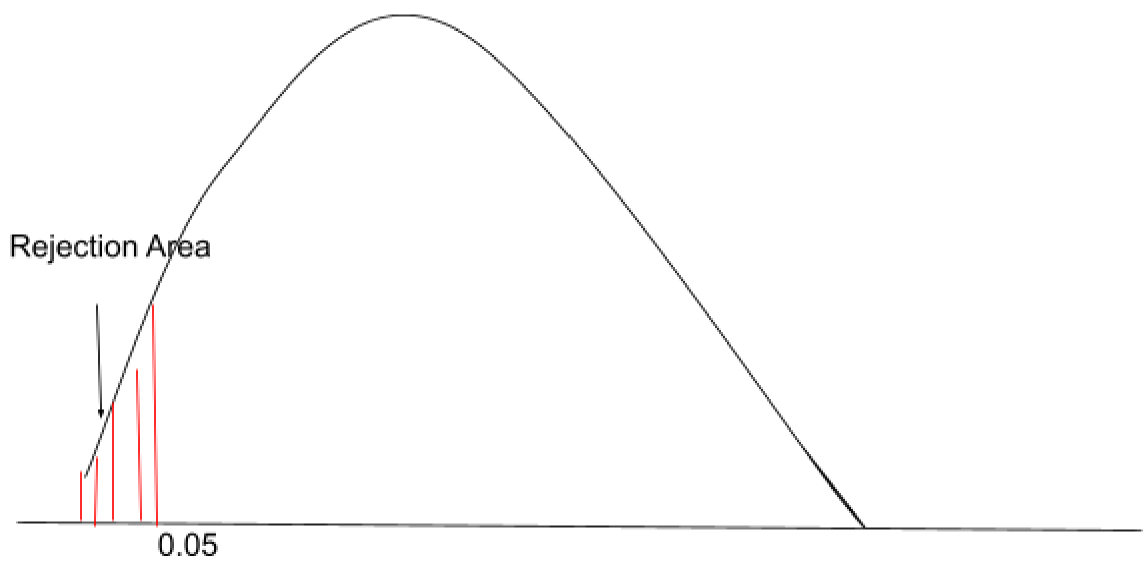
Z-score = (X-bar - μ0)/ (sd/sqrt(n))

Z-score = (0.8 - 0.6)/ (2.5/sqrt(1000))

Z-score = 0.2 / 0.0.06

Z-score = 2.5

Now that we have z-score, lets compute area under the curve from z-score table is 0.9938.



Hence p-value = area on the left = 1-0.9938 = 0.0068

This p-value of 0.0068 < alpha (0.05), **hence we reject the null hypothesis.**

Thus we proved that the RAG system performs better than the bm25 system

Quantitative Results:

I have done an evaluation on 2000 long and short queries spread across 23 users.

NDCG@1 Evaluation:

In order to measure NDCG@1 I have captured relevance score (1-4 1 being lowest and 4 being highest) along with position (1) for each combination of query and document. I have measured this score for RAG system as well as OKAPI (bm25) system.

The format of raw result is as follows

UserId	Query	Docid	RelevanceScore (RAG)	Relevance Score (Okapi BM 25)	NDCGScore (RAG)	NDCGScore (Okapi BM 25)
121	What are the implications of quantum computing on encryption?	10031	3	1	0.74	0.51
109	How does the theory of relativity impact modern physics?	10179	4	4	0.65	0.53
…	….	….	….	….	….	..

Where

- UserId: Unique id for the user assigned to do the evaluation. There are 23 unique users.
- Query: Search Query
- Docid: UNique id of document surfaced in the search result

- Relevance Score (RAG): User's ratings (1-4 scale) indicating how relevant the document is for the given query for RAG system
- Relevance Score (bm25): User's ratings (1-4 scale) indicating how relevant the document is for the given query for BM25 system
- NDCG@1 (RAG): This score is computed from RAG relevance score and using NDCG equation 2 with K=1
- NDCG@1 (bm25): This score is computed from bm25 relevance score and using NDCG equation 2 with K=1

There are 2 excel files which contains raw evaluations.
NDCG@1 Long queries evaluations : download link
NDCG@1 short queries evaluations : download link

Precision@1 Evaluation:
In order to measure Precision@1 I have captured precision score (0-1 1 being True Posistive and 0 being False Positive) for each combination of query and document. I have measured this score for RAG system as well as OKAPI (bm25) system.
The format of raw result is as follows

UserId	Query	Docid	Precision@1 (RAG)	Precision@1 (Okapi BM 25)
112	What are the implications of quantum computing on encryption?	10085	1	0
120	How does the theory of relativity impact modern physics?	10196	1	0
….	…	…	…	…

Where

- UserId: Unique id for the user assigned to do the evaluation. There are 23 unique users.
- Query: Search Query
- Docid: UNique id of document surfaced in the search result
- Precision@1 (RAG): This score is computed as described in Precisiob@k equation 1 with K=1
- Precision@1 (bm25): This score is computed as described in Precisiob@k equation 1 with K=1

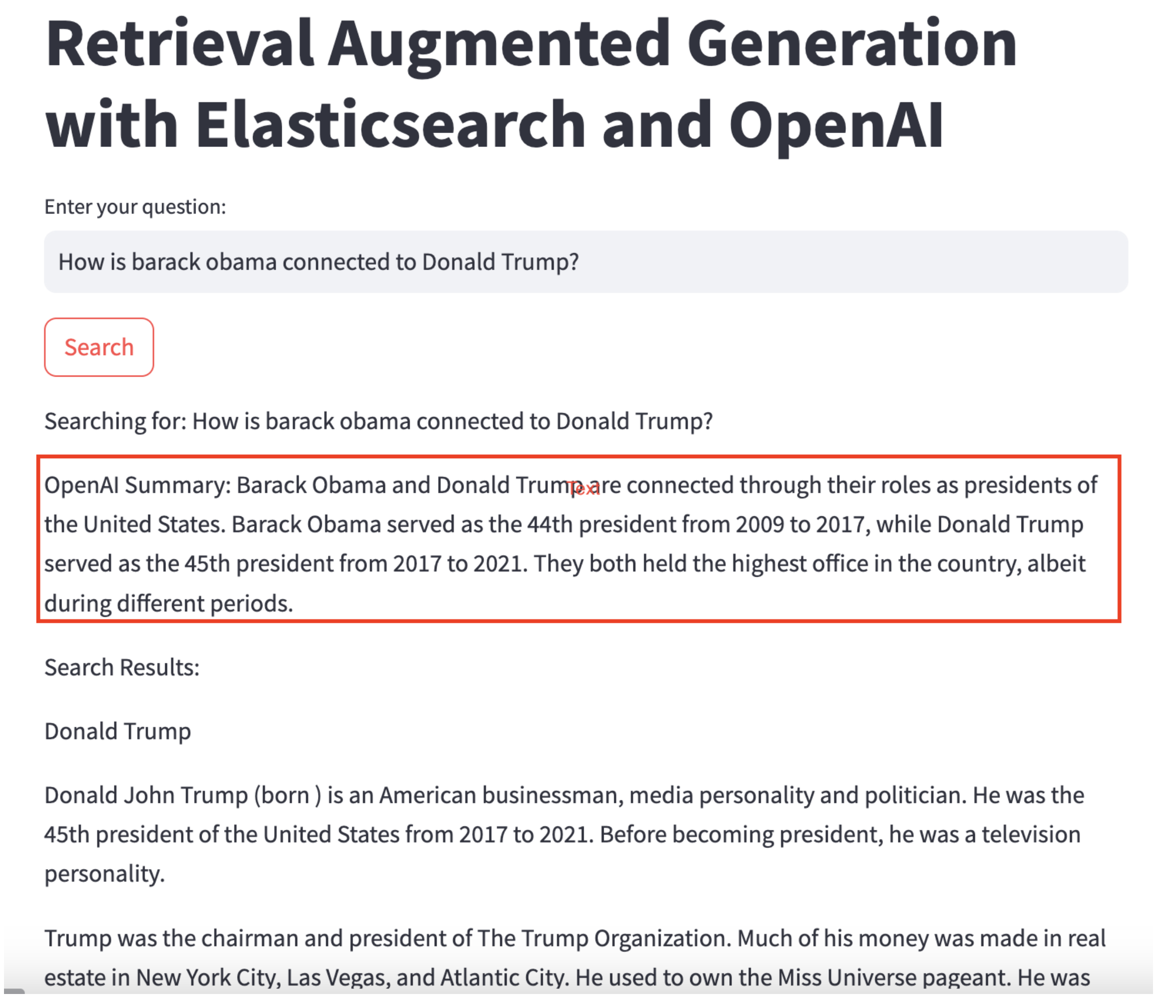
There are 2 excel files which contain raw evaluations.
Precision@1 Long queries evaluations: download link
Precision@1 short queries evaluations: download link

Aggregated Results:
Here are aggregated mean score results from raw excel file.

Metric	Long Queries (> 3 tokens)		Short Queries (< 3 tokens)	
	OKAPI BM25	**RAG**	**OKAPI BM25**	**RAG**
Precision@1	0.28	0.73 (Winner)	0.7065 (Winner)	0.29
NDCG@1	0.28	0.74 (Winner)	0.64 (Winner)	0.41

As you can clearly see from the above table for longer queries, Precision@1 and NDCG@1 for RAG system is higher than OKAPI BM25 system. In contrast, for shorter queries Precision@1 and NDCG@1 for RAG system is lower than that of OKAPI BM25 system.

Aesthetic Score:
The Aesthetic score measures the overall system's appeal to the user regarding ease of use and ability to nail the intent. It compliments NDCG@1 and Precision@1 with overall experience in terms of usefulness. We added a summary of results in the case of RAG in addition to results which we hope can reduce the need for the user to fulfill his tent.
Here is the screenshot that shows a summary of all search results at the top.



Retrieval Augmented Generation with Elasticsearch and OpenAI

Searching for: How is barack obama connected to Donald Trump?

OpenAI Summary: Barack Obama and Donald Trump are connected through their roles as presidents of the United States. Barack Obama served as the 44th president from 2009 to 2017, while Donald Trump served as the 45th president from 2017 to 2021. They both held the highest office in the country, albeit during different periods.

Search Results:

Donald Trump

Donald John Trump (born) is an American businessman, media personality and politician. He was the 45th president of the United States from 2017 to 2021. Before becoming president, he was a television personality.

Trump was the chairman and president of The Trump Organization. Much of his money was made in real estate in New York City, Las Vegas, and Atlantic City. He used to own the Miss Universe pageant. He was

Evaluation:

To measure aesthetic score, A dataset was created a set of 2000 queries and divided it among 100 unique users. The dataset was distributed among 100 users and asked them to compare the overall aesthetics of the RAG system and BM25 system and give one of the 4 ratings for long (> 3 tokens) and short queries (<3 tokens) separately.

- RAG is better than BM25
- BM25 is better than RAG
- Both are good
- Both are bad

The raw results are collected in excel sheet and are available at locations below.

Long Queries Results: download link

Short Queries Result: download link

I then aggregated each bucket of rating and computed overall effectiveness as shown below,

Aggregated Result

Metric	**Aesthetic Score**	
	Long Queries (> 3 tokens)	**Short Queries (< 3 tokens)**
Both Good	10%	10%
Both BAD	5%	0%
RAG is Better	60%	15%
OKAPI BM25 is better	25%	75%

As you can see from the above table for longer queries, Aesthetic score for RAG system is higher than OKAPI BM25 system whereas for shorter queries Aesthetic score for RAG system is lower than that of OKAPI BM25 system

CONCLUSION:

We found that NDCG@1, Precision@1 and Aesthetic score is better for RAG than bm25 for long queries (> 3 tokens). At the same time, we also see BM25 system scores much better on NDCG@1, Precision@1, and Aesthetic scores than RAG for short queries (< 3 tokens).

Users are more satisfied with one summarized answer than browsing through N different links to get to the final answer. This dramatically improved the productivity of the user.

RAG algorithm fails on short queries as context is not enough to nail the answer. Hence final system would be a hybrid system where bm25 continues to work well for short queries, and RAG can be leveraged for long queries.

We also found that although the quality of search results went up, the system became more complex and more expensive. Also, we added dependency on external systems like OpenAI.

Future Work:

- We leveraged OpenAI API, which charges consumers based on several tokens in the request and response. The cost can add up over time . To control the cost, one can decide to use commercial open-source llama-30B large language models and host on a corporate GPU farm.
- We could perform similar evaluations on Latin and Asian languages as well to see if we see similar results.

REFERENCES

[1] Lewis, Patrick, et al. "Retrieval-augmented generation for knowledge-intensive NLP tasks." Advances in Neural Information Processing Systems 33 (2020): 9459-9474.

[2] Gao, Linhao, Xiang Deng, and Qingkai Zeng. "RETRO: Efficient Retriever-Based Open-Domain Dialogue System." arXiv preprint arXiv:2112.05238 (2021).Ddd

[3] Fabbri, Alexander R., et al. "Multi-Encoder Architectures for Retrieval-Augmented Generation." arXiv preprint arXiv:2110.08329 (2021).

[4] Ouyang, Leo, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chandra Bhagavatula, Andrew Dai, and Pranav Shyam. "Training Language Models to Follow Instructions with Human Feedback." arXiv preprint arXiv:2203.02155 (2022).

[5] Lee, Kenton, Ming-Wei Chang, and Kristina Toutanova. "Learning to summarize from human feedback." arXiv preprint arXiv:2109.08306 (2021).Dd

[6] Zhang, Saizheng, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. "Recipes for Building an Open-Domain Chatbot." arXiv preprint arXiv:2004.13637 (2020).Dd

[7] Chen, Wenhu, Wang-Chiew Tan, Xavier Tannier, and Vincent Y. Wang. "Tabgren: Table-aware bert for tabular data-to-text generation." arXiv preprint arXiv:2007.06827 (2020).

[8] Chen, Wenhu, Wang-Chiew Tan, Xavier Tannier, and Vincent Y. Wang. "Tabgren: Table-aware bert for tabular data-to-text generation." arXiv preprint arXiv:2007.06827 (2020).

[9] Borgeaud, Sebastian, et al. "Improving text generation with student-forcing optimal transport." arXiv preprint arXiv:2010.07574 (2020).

[10] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... & Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461.

[11] Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., & Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. arXiv preprint arXiv:1611.09268.

[12] Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., & Riedel, S. (2019). Language models as knowledge bases?. arXiv preprint arXiv:1909.01066.

[13] Zhang, Y., Sun, S., Galley, M., Chen, Y. C., Brockett, C., Gao, X., ... & Dolan, B. (2018). QuAC: Question answering in context. arXiv preprint arXiv:1808.07036.

[14] OpenAI. 2023. GPT-4 Technical Report. arXiv:cs.CL/2303.08774

[15] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.