Building a Retrieval-Augmented Generation (RAG) System for Academic Papers

Report Prepared by Artashes Mezhlumyan
in part fulfillment of the degree requirements
for the Degree of BS in Data Science
in the Zaven and Sonia Akian College of Science and Engineering
*Supervisor: Anna Grigoryan *Coordinator: Habet Madoyan

***Abstract*—This report presents the final results of our capstone project, which focuses on developing a Retrieval-Augmented Generation (RAG) system designed for navigating through the vast amount of academic papers. The Retrieval-Augmented Generation (RAG) system enhances search capabilities by integrating search strategies for retrieving data and LLM models for generating text, addressing the limitations of traditional search engines like Google, which may struggle with interpreting complex, scholarly queries and providing contextually relevant academic insights. Our proposed RAG system seeks to address these challenges by leveraging advanced techniques in document retrieval and natural language processing to offer precise, contextually relevant excerpts in response to user queries. The system utilizes a 2-step vector search using the vector search with cosine similarity metric on an HNSW index on the paper's abstracts and the papers itself to pass only relevant information to LLM; this enables enhanced data retrieval and contextually aware text generation. This report shows our achievements in implementing various system components, including document retrieval, search methods, text generation, and initial performance evaluation. We experimented with a number of search strategies for knowledge retrieval, found our best-performing RAG search style, experimented with a number of LLMs, and made the final RAG system. We also discuss the encountered limitations, insights gained, and potential avenues for further improvement.**

I. Literature Review

A. Overview

The retrieval-augmented Generation (RAG) system is initially designed to overcome the limitations of traditional search engines like Google. Sometimes, search engines fail to interpret complicated queries, returning unrelated information to the user. However, the RAG system enhances search capabilities by retrieving relevant information to the user's query and passing it to LLM, which generates contextually relevant text responses.

B. How RAG system works

Retrieval-augmented generation has a two-step operation process: retrieval and generation. Firstly, a vector search is conducted on data using similarity metrics. The first step quickly filters relevant information about users' queries and passes it to the chosen Large Language Model. Once relevant documents are retrieved, the generation phase begins. The chosen Large Language Model generates answers to queries based on retrieved data and returns comprehensive and coherent answers.

C. Applications of RAG Systems

1) Customer Support: Retrieval-augmented generation systems are widely used in the Customer Support field. RAG system uses customer data to give personalized responses based on purchase history, location, past issues, and communication preferences. This kind of RAG application is widely used in production. Vivid examples are Clinc and Kasisto, which deliver exceptional automated service across their channels.

2) Automatic Content creation: RAG empowers to generate posts for social media blogs or webpage content using custom knowledge bases. Tools like Quill, Writesonic, and Wizdom use RAG to create creative content.

3) Document Summarization: RAG can efficiently summarize the most essential or salient point by combining text and transcripts. Apps like Otter or Krisp already use RAG to highlight critical talking points from meetings.

4) Recomendation Systems: RAG takes recommendation engines to the next level by scanning customer data to give ultra-relevant suggestions. Platforms like Spotify and Overcast explore RAG to factor taste profiles into suggestions.

D. In-depth Examination of the RAG System

Beyond the practical applications, the academic research around Retrieval-Augmented Generation (RAG) systems offers significant insights into their potential and limitations. A pivotal study titled [1]Retrieval-Augmented Generation for Large Language Models: A Survey reviews the evolution of RAG systems. They highlight how RAG addresses LLM's traditional challenges, such as hallucinations or outdated knowledge. Integrating the RAG system with an external database for specific tasks enhances accuracy and credibility and allows continuous knowledge updates with domain-specific integrations. The paper also provides a comprehensive overview of RAG architecture, from simple implementation to advanced approaches.

Similarly, a study conducted by Patrick Lewis and Ethan Perez, named [2]Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, presents the application of RAG in open-domain Question-Answering tasks, setting new benchmarks against conventional seq2seq and task-specific architectures. The study represents how RAG can be optimized to produce more specific and diverse applications, considering all the LLMs' issues.

Moreover, technical advancements in vectorization and handling semi-structured data are discussed in [3]A Method for Parsing and Vectorization of Semi-structured Data used in Retrieval Augmented Generation study. The paper represents a method that significantly enhances the precision, reliability, and diversity of LLM outputs. It also introduces how to work with diverse document formats and make a structured database for RAG retrieval.
These studies underscore how RAG pushes the boundaries of traditional LLM text generation and helps to create more dynamic, accurate, and user-specific applications.

E. Conclusion

RAG system effectively combines retrieval techniques and text generation, offering robust solutions for various information-intensive applications. Outperforming traditional search engines and being widely used in production applications. Taking into account all these above-mentioned points, the RAG system has a lot of promises and is worth diving deep into.

II. Data collection

We have identified an up-to-date dataset hosted on Kaggle by Cornell University, named [4]"arXiv dataset". This dataset includes comprehensive details about academic papers on arXiv.org, including paper IDs, abstracts, authors, and more. The 'arXiv dataset' includes over 2 million academic papers in math, statistics, electrical engineering, quantitative biology, economics, and more. This diversity and volume ensures that our system has a strong foundation for retrieving and generating content." This discovery is critical for our project, as it provides a rich source of current academic content, enabling our RAG system to operate with the latest research papers. Utilizing this dataset will significantly enhance our system's ability to deliver relevant and timely academic information. Moreover, We implemented Python code for pdf retrieval, which uses the Python "requests" library to send requests to arxiv.org with appropriate paper IDs and download the desired pdf. In this manner, We have downloaded the whole pdf dataset from arxiv.org.

III. Methodology

A. Overview of Methodology

This section outlines the methodology used to develop our RAG system, focusing on data retrieval, preprocessing, semantic search, and optimization techniques to handle large-scale academic data efficiently. As we are using LLMs for our system, we have input limitations and need to input to the model limited text, which is related to the user's query. Here, RAG comes into use. In the Retrieval-Augmented Generation (RAG) system, we emphasize using search functionalities to efficiently navigate and extract relevant information from a vast corpus of academic papers. The next step is to find an efficient search strategy for our RAG system.

B. Data Retrieval and Preprocessing

1) Keyword Search: We initiated our experiments by focusing on keyword searches and leveraging the rich dataset to test and refine our document retrieval component. This step is critical in understanding the dynamics of efficient information extraction and sets the stage for more complex retrieval and generation tasks ahead. During our experiments, we found that the limitations of keyword search are a significant burden for our RAG system as keyword search does not always work precisely. For example, We implemented the Natural Language Toolkit (NLTK) Python library for keyword search and got the following results.

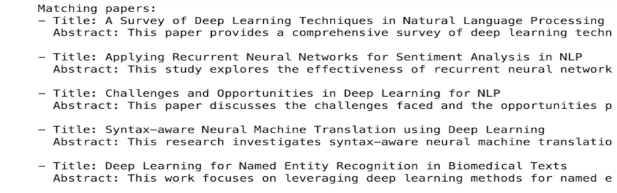

Fig. 1. When searching for "deep learning in natural language processing," We got pretty good results.

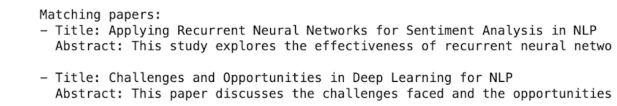

Fig. 2. However, We get worse results when We shorten my query to "deeplearning in NLP," which has the same meaning as the last query.

Initially, a keyword search was employed to establish a baseline for document retrieval efficiency. However, its limitations became apparent, as it often needed to capture the context of queries, leading to inaccurate or irrelevant results. We bumped into the limitation of simple keyword search and decided to give a chance to other search methods. These challenges necessitated a transition to semantic search, a more advanced method that interprets the underlying intent of queries through natural language understanding.

2) Transition to Semantic Search: Building on our initial experiments, we transitioned to semantic search, known as vector search, to enhance the sophistication and effectiveness of our retrieval process. Instead of traditional keyword matching, semantic search understands the contextual meaning and the intent behind a user's query. Instead of relying solely on the exact words, semantic search uses advanced natural language processing (NLP) techniques and machine learning algorithms to analyze and interpret the query's context. It then maps this query to a high-dimensional space where the query and the documents in the search index are represented as vectors. The semantic relationships between words and phrases are encoded in these vectors, enabling the search system to retrieve documents that are contextually related to the query, even if they don't contain the exact words. Vector search (semantic search) involves converting text data into high-dimensional vectors representing the words' semantic meaning, enabling

the system to perform searches based on conceptual similarity rather than mere keyword matching.

C. Semantic Search Implementation

1) Vector Embeddings: To implement semantic search, we first convert text data into numerical representation. There are various approaches for text embeddings, starting from Word2vec and continuing with more advanced techniques. We experimented with the Word2Vec model in our early stages of data vectorization. Word2Vec, a pioneering vector embedding model, is adept at capturing semantic and syntactic word relationships but falls short of comprehending the full scope of sentence-level or paragraph-level context. This limitation is crucial in academic settings where precision and nuanced understanding are paramount. We went beyond that and decided to use transformer models for this task. We employed the SBERT model [5]"multi-qa-MiniLM-L6-cos-v1" for vectorizing our data, leveraging its capacity to understand and encode the nuanced context of academic texts. The model is based on a distilled version of BERT, which is significantly smaller and faster. This makes it more practical for real-time applications. Also, This model is fine-tuned explicitly for question-answering and semantic similarity tasks, making it highly effective at understanding and encoding nuanced academic texts. Despite its strengths, this approach faced limitations, particularly concerning token limits, which could restrict the depth of text analysis. A token is a unit of text that the model processes, and our chosen model can process 512-word pieces at once. We have many cases where the text is longer than that size. We addressed this challenge by calculating the mean vector of our abstract vectorizations, effectively summarizing the semantic essence of texts while bypassing token limitations. This strategy allowed us to maintain the integrity of semantic search while accommodating the extensive content of academic papers. We saved the vectors in CSV format. Each entry in the CSV file contains the vector representation of an abstract alongside the original abstract text and the paper's unique ID. This approach enables efficient retrieval and utilization of information during the vector search, allowing for quick and accurate matching of user queries with relevant academic content.

D. Implementation of Semantic Search

When a user asks a question, our RAG system transforms it into vectors and measures the similarity with our vectorized knowledge base through our selected model. The program then seamlessly integrates the matched information with the user's query, forwarding this context to our large language model. Employing semantic search enabled the RAG system to align more closely with user queries by understanding contextual relationships within the data, significantly enhancing retrieval accuracy.

E. Search Optimisation

1) Initial Vector Search: Naiv implementation of vector search is scalable as its calculation of cosine similarity of query and each data point for every search attempt; this naive search approach can't be implemented at a billion scale, leading us to explore vector indexing solutions - to save and search vectors efficiently. An efficient solution for the billion scales is the vector index. A vector index employs specialized data structures (such as approximate nearest neighbor (ANN) techniques like Locality-Sensitive Hashing (LSH) and HNSW) to organize the vectors in a way that accelerates the search process. The critical difference lies in performance and scalability: naive vector search is simple but slow. It scales poorly with large datasets, while vector indexing introduces complexity in data organization to achieve much faster search times, making it viable for large-scale and real-time applications. An exceptionally compelling illustration of vector indexing can be found [6]here. We got our hands dirty and started experimenting with different libraries. We've conducted extensive experimentation with the SBERT library, utilizing cosine similarity for semantic search, and explored the FAISS library by implementing a FlatIndexL2 search. Additionally, we experimented with the HNSW library, employing searches based on cosine similarity, Inner Product (IP), and L2 distances. To further enhance our system's capabilities, we experimented with combining these search methodologies through hybrid search methods, aiming to leverage the strengths of each approach for improved performance and accuracy in retrieving relevant academic content.

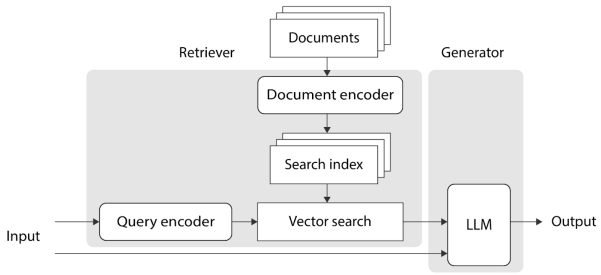

Fig. 3.

2) Advancements in Vector Search (Use of HNSW): HNSW emerged as the optimal method due to its search speed and accuracy balance. Compared to other methods like FlatIndexL2, HNSW provided faster response times without sacrificing accuracy based on our primary validation of anecdotal examples.

Now that we have all the pieces for the RAG system, our next step is to integrate them seamlessly.

F. Two-Layer Vector Search

1) Handling Large Text Data: We are searching for an abstract level to find which papers are relevant to users' queries. As we discussed, we are implementing the cosine similarity search (using the HNSW library) on our abstracts of papers, which returns relevant papers ' IDs. Like in all Retrieval-Augmented Generation applications, the next step is to pass the paper's text to the chosen LLM along with the

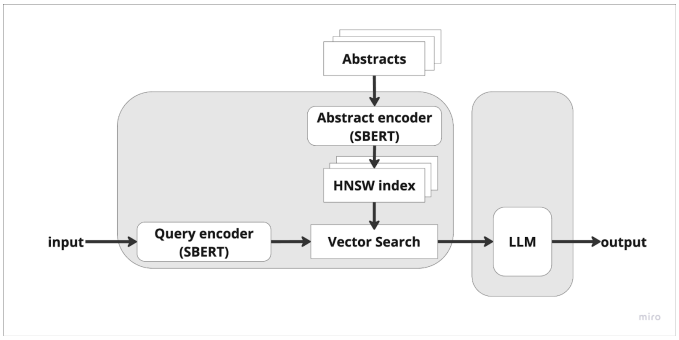

Fig. 4.

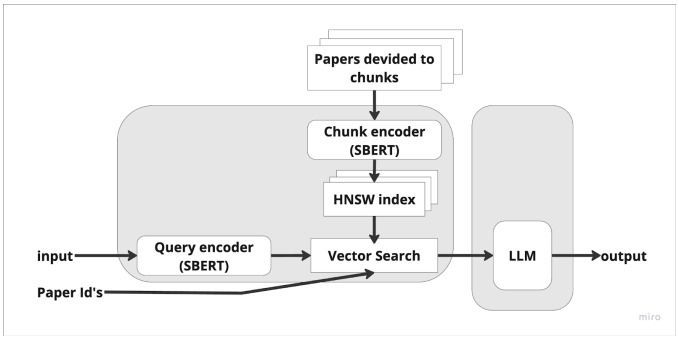

Fig. 6.

user's query. However, academic papers are pretty long in text size; sometimes, only one small paragraph is relevant to the user's question. Also, it is not always a good idea to input as much text as possible to the Large Language Model as it can hallucinate or generate answers that are out of context. For example, one of the best models today - 'gpt-3.5-turbo-0125' has a maximum input of 16,385 tokens per request (one of the biggest, if not the most significant input size today); in our case, the input can fit a maximum of 3 papers, which sometimes can be not enough. It would be better to fit ten papers but with the top 5 the most relevant chunks of text from each paper. We implemented a two-layer search system to address the token limitations of large language models. The first layer identifies relevant papers, while the second narrows down to the most pertinent text excerpts, ensuring focused inputs to the language model.

2) Optimising Search for Relevance: So, we decided to add a second layer of search. We implemented vector search, which compared the user's query to chosen pdf chunks, which outputs top similar chunks from our pdfs. This two-layer search method optimizes handling large text volumes and enhances the system's accuracy in providing highly relevant responses, which is particularly effective for complex academic inquiries. See Fig.5 and Fig.6:

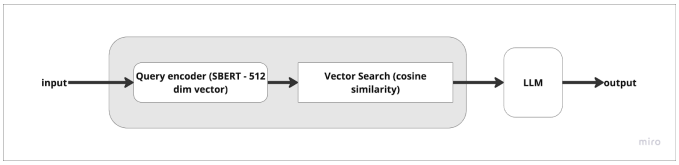

Fig. 5.

IV. Evaluation of Vector Search

We generated a synthetic dataset of questions to understand how well our Vector Search works. We generated a synthetic dataset resembling typical user queries to simulate real-world applications. This approach allowed us to control the variability and complexity of the queries, ensuring a robust test of our system's capabilities. We input each pdf document with proper instructions into the gpt-3.5 model APWe to generate these questions and generated 200 synthetic sample questions. Initially, We entered each question individually into our RAG system to see if it could identify relevant papers related to those queries. In evaluating vector search, we used precision-at-k (where k=3) as a metric, noting that in 199/200 cases, the correct paper appeared in the top 3 results. This demonstrates a precision rate of 99.5%, significantly above industry benchmarks. Further investigation is needed for the outlier to understand the discrepancy. Is the information in the paper's abstract different from the main topics discussed in the paper, or does the abstract not accurately reflect the content of the paper? These questions led to further inquiries, suggesting that the initial search performance met our expectations.

Going forward, we know that the First layer of our RAG system works well; however, do paper abstracts contain enough information about full papers? Are they similar to full documents, or are abstracts descriptive enough to rely on them? We computed cosine similarity scores across segmented text chunks to assess the representativeness of abstracts versus full papers. These were then averaged to derive a mean similarity score per paper, which was visualized alongside a confidence interval to assess the normal distribution of the data We have used our abstract embeddings and PDF embedding chunks. After calculating the similarity of PDF chunk by chunk to abstract and then taking the average similarity of each PDF document, We visualized the results.

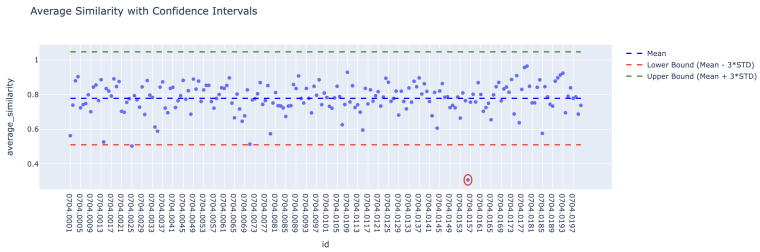

Fig. 7.

From an eyesight view, the papers are similar to abstracts; however, We have drawn confidence interval lines using empirical rules to understand our visualization results better. Confidence intervals were calculated using the empirical rule to determine if the similarity scores were normally distributed.

This statistical approach highlighted outliers, suggesting cases where abstracts may not adequately summarize the full papers. Such findings are crucial for refining our indexing strategies; they tell us whether most values lie in a normal distribution. Analyzing the distribution of paper vs abstract contextual relatedness, the abstracts cover most of the context of the paper. We can see that only one pdf document is far from the confidence interval lines. After carefully examining this document, We see a big vocabulary difference between the abstract and the paper. However, our vector search successfully found this paper anyway.

V. Chosen LLM's

To evaluate the results of our RAG system, we decided to use several large language models for more diverse results. We decided to take [7]'gpt-3.5-turbo-0125' from the top of industry models and started to look for its competitors. The main competitors of OpenAWe models are LLM's made by Google and Meta. We chose [8]'Gemini-pro' from Google's model collection, and we took [9]'Meta-Llama-3-8B' from Meta. For more widespread analysis, we selected [10]Cohere as a fast-growing, famous LLM and considered [11]'google-flan-t5' one of the famous retired LLM models.

VI. Evaluation

We generated 100 sample questions to evaluate our models, inputting each pdf document with proper instructions into the gpt-4 model. Implementing our RAG system with every chosen LLM, We inputted 100 sample questions for every LLM case and documented the results.

Many famous evaluation metrics and algorithms are in the market, from truth measurement multitasking evaluation to sentence completion evaluations. However, in our case, we first need to understand how informative or truthful LLMs are. First, we need to be sure that the question is being answered from the correct information source. From then on, we decided to search metrics that would provide us with informativeness, instructiveness, or explanatory scores. In evaluating the performance of a Retrieval-Augmented Generation model on Large Language Models (LLMs), metrics were carefully chosen to assess different aspects of content quality. Information Density was calculated by counting non-stopword tokens per 100 words to gauge content richness. Subject Proportion was measured using SpaCy's POS tagging to determine the focus, and NER Proportion utilized SpaCy's NER to track the relevance and presence of critical entities in the responses.

A. Information Density

Firstly, Information Density was calculated by normalizing the count of unique non-stop words by the total number of words, measuring how much unique information the model generates per 100 words.

B. Subject Proportion

The Subject Proportion metric was established by identifying subject words using part-of-speech tagging with SpaCy. This involved calculating the proportion of unique subject words to the total number of words in the response, offering insights into the subject focus and variability in the text.

C. NER Proportion

Lastly, the NER Proportion was determined by applying Named Entity Recognition (NER) to identify entities in the text and computing the proportion of these entities relative to the total word count. This metric helped me understand how well the model captures and generates critical entities within the response context. Together, these evaluation techniques allowed for a comprehensive analysis of the model's output, facilitating a detailed comparison with other models to gauge performance efficacy in various aspects.

1) GPT Evaluation:

a) Information Density: Starting with the most widely-used language model family today, GPT, we find that the information density in the results is moderate, typically ranging from 30% to 60%. This suggests that the majority of responses possess a medium level of detail. Regarding the density of abstract information, our measurements show that it is slightly less informative than the responses from GPT, indicating that GPT generally performs well.

b) Subject Proportion: For Subject proportion, most responses are concentrated around a proportion of 5%, suggesting a focused but not overly detailed approach to subjects. However, seeing a similar pattern for abstract subject proportion, GPT fulfilled our expectations.

c) NER Proportion: The GPT model's responses generally contain fewer named entities (like names of people, organizations, or locations) than other parts of the content. Outliers indicate some responses where the model includes a significantly higher number of named entities, but such responses are exceptions rather than the rule. Again, we see that the abstract's NER proportion has the same trend as the Answer's NER proportion, showing that as many named entities were present in the abstracts, the same number of named entities were present in GPT's response.

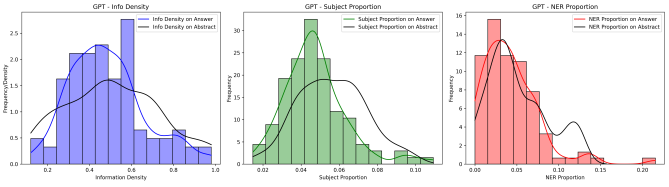

Fig. 8.

2) Cohere Evaluation:

a) Information Density: Continuing with the analysis of the Cohere model, the information density results are notably lower than those of GPT, peaking at 30%. Furthermore, the density of abstract information reaches up to 50%, suggesting

that Cohere's responses generally contain a lower level of information density.

b) Subject Proportion: However, the proportion of Cohere's answers peaks at 3% to 7%. These numbers indicate that the model concentrates on the main subjects within a narrow range. The Abstract's subject proportion repeats the same trend, indicating that the model answered within the same subject range. This suggests a certain level of specificity and consistency in addressing the central topics within the responses.

c) NER Proportion: Like GPT, Cohere has close results for NER proportion, indicating that this model tends to generate few named entities, as it repeats papers' named entities and does not go off the topic.

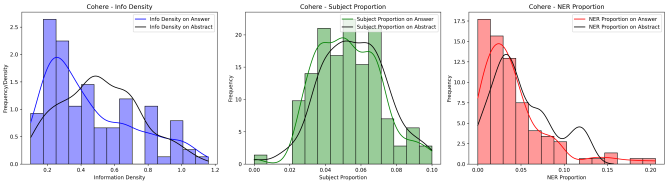

Fig. 9.

3) Flan-T5 Evaluation:

a) Information Density: Discussing the vintage 'google-flan-t5' model, it underperforms across all metrics. Yes, information density ranges up to 20%, but it is far from abstract info density. It tells us that T5's answers have very different info Density than the papers.

b) Subject Proportion: Subject proportion peaks at 0 to 10%, where the abstract subject proportion should peak. However, some outliers tell that the model can get off the track.

c) NER Proportion: The ner proportion trend also has outliers, showing that the model used entities named differently than the abstract.

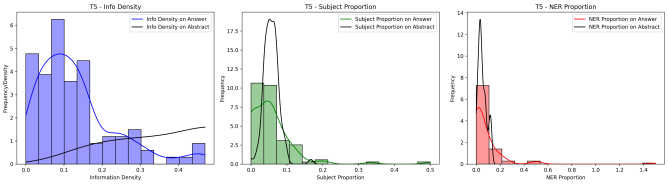

Fig. 10.

4) Gemini Evaluation:

a) Information Density: Returning to today's competitors in the LLM world, we can look at 'Gemini-pro' results. As we know, Gemini is the main competitor of the GPT models family, and We will compare them. Unfortunately, Gemini has low info density, peaking from 10% to 20%. Unfortunately, abstracts' information density has a different trend and does not match Gemini's answer density. Anyways, this is significantly lower than the GPT results.

b) Subject Proportion: However, the subject proportion of Gemini copies the trend of the abstract's results, showing that the exact subject words were kept as in the original abstract. It's worth mentioning that in this aspect, the Gemini had nearly the same results as GPT, which was unexpected. Sorrowfully, Gemini-generated answers could have been better in Named entities, generating answers containing many named entities or being too generous and getting out of context named entities.

c) NER Proportion: Gemini has a low NER proportion and a lower distribution than GPT or Cohere, indicating a less frequent mention of named entities. Gemini's performance falls short when measured against GPT, which has taken the lead in this comparison.

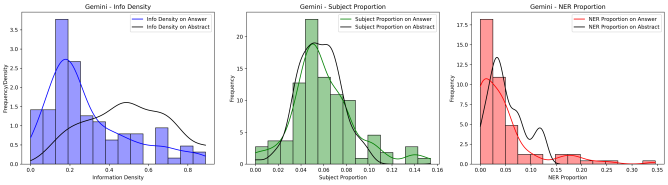

Fig. 11.

5) Llama3 Evaluation:

a) Information Density: Finally, we tested 'llama3' as a product from another industry giant; the outcomes were unfortunately underwhelming. The information density in the model's answers ranged up to a mere 0.40%(far away from abstract information density results). It reached its top near 0, still worse than the older 'google-flan-t5' performance.

b) Subject Proportion: Likewise, the subject proportion hovered close to zero, far from the abstract's results.

c) NER Proportion: In the same manner, we have a lot of outliers for the NER proportion. Initially, our abstracts were not reached in their NER proportion results, and we expect to see the same results in LLM answers as they should stay on the topic and generate new words, which was not the case for Gemini. It went off the track in all positions.

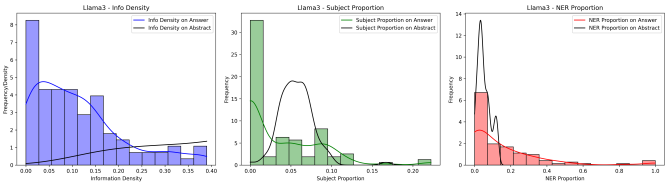

Fig. 12.

D. To Conclude

To conclude, the GPT model had the best overall performance. It showed moderate information in its answers and a good focus on the main subjects, without too many details. It didn't often use named entities, but its competitors did not stand out with other metrics. The model generally coped well with the paper's information density, subject proportion, and

paper-named entities. GPT outperformed other models in balancing detail and clarity, particularly regarding subject focus and information density. This suggests that GPT's algorithms are better suited for applications requiring nuanced content generation, such as academic or technical writing, where detail and accuracy are paramount.

VII. Scalability and Limitations

Exploring the scalability of various large language models (LLMs) that we have chosen could potentially be integrated into our Retrieval-Augmented Generation system. We assess their performance based on the context window size, inference time, cost efficiency, API availability, and model parameters. The following table summarizes the capabilities and limitations of each model, providing insights into their suitability for academic search and generation tasks:

	Context Window	Inference Time	Cost per 1M token input	Cost per 1M token output	Api availability	Model parameters
Gpt-3.5-turbo-0125	16,385 tokens	2.4 sec	$0.50	$1.50	Yes	175B
Cohere-command-r	128,000 tokens	10.4 sec	$0.50	$1.50	Yes	35B
Gemini-pro	128,000 tokens	2.9 sec	7$	21$	Yes	50T
Meta-Llama-3-8B	8,200 tokens	0.5 sec	0$	0$	No	8B
Google-flan-t5-base	512 tokens	0.23 sec	0$	0$	No	250M

Fig. 13.

Performance Without Our RAG System To evaluate the raw performance of each LLM further, we measured the end-to-end response times:

- GPT-3.5-Turbo-0125: 4.15 seconds
- Cohere-Command-r: 8.72 seconds
- Gemini-Pro: 8.71 seconds
- Meta-Llama-3-8B: 23.17 seconds
- Google-Flan-T5-Base: 5.69 seconds

Discussion The performance metrics show a significant variance in terms of both cost and efficiency. Models like Meta-Llama-3-8B and Google-Flan-T5-Base offer free usage but are limited by smaller context windows, and lack of APWe availability may hinder their scalability in a production environment. On the other hand, Gemini-Pro provides a large context window and quick response time but at a higher cost, making it less cost-effective for extensive querying. Models like GPT-3.5-Turbo-0125 and Cohere-Command-r offer a balanced approach with reasonable cost and performance metrics, making them suitable candidates for integration into our RAG system, especially considering their large context windows and APWe availability, which are crucial for handling complex academic queries.

Our RAG system is designed with a two-step retrieval process. The first layer involves a vector search on abstracts, which takes approximately 0.033 seconds per query. Following the initial filtering, the second layer of our RAG system conducts a more detailed search through the selected documents. This second layer of search is executed in just 0.191 seconds. Together, these two layers ensure that our system not only quickly identifies relevant documents but also extracts the most pertinent information from those documents, making it highly scalable for handling numerous queries in an academic setting.

The performance of the two-step retrieval process in our RAG system is significantly influenced by the characteristics of the selected language models (LLMs). For instance, the large context window of Cohere-Command-r enables more comprehensive initial searches in the first layer, which reduces the load on the second layer of detailed search. This not only improves the efficiency of the system but also ensures that the most relevant documents are prioritized early on. Also, the availability of APIs for models like GPT-3.5-Turbo-0125 facilitates seamless integration into the system, enhancing the automation of the retrieval process. This APWe connectivity allows for dynamic updates and handling of complex queries, which is crucial for academic research environments.

VIII. Conclusion

In conclusion, the development and evaluation of our Retrieval-Augmented Generation (RAG) system demonstrates its potential to revolutionize the retrieval and analysis of academic papers. By effectively integrating advanced large language models with our innovative two-layer retrieval technique, our system addresses the complexities of academic search with high efficiency and accuracy. Our suggested two-layer vector search system ensures that RAG not only identifies relevant documents correctly but also processes them to extract the most relevant information. This capability makes it particularly valuable in academic settings where speed and precision are crucial. Looking forward, the continuous refinement of our system's components, including the integration of more sophisticated LLMs and the optimization of retrieval algorithms, will further enhance its performance and scalability. This project lays a solid foundation for future advancements in academic information retrieval, promising to support researchers by providing quicker, more accurate access to academic literature.

References

[1] Y. X. Yunfan Gao, "Retrieval-augmented generation for large language models: A survey," https://arxiv.org/abs/2312.10997v5, 2023.

[2] E. P. Patrick Lewis, "Retrieval-augmented generation for knowledge-intensive nlp tasks," https://arxiv.org/abs/2005.11401, 2020.

[3] J. G. Hang Yang, "A method for parsing and vectorization of semi-structured data used in retrieval augmented generation," https://arxiv.org/abs/2405.03989, 2024.

[4] "arxiv dataset," https://www.kaggle.com/datasets/Cornell-University/arxiv, accessed: 2024-05-07.

[5] "sentence-transformers/multi-qa-minilm-l6-cos-v1," https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1.

[6] "Hierarchical navigable small worlds (hnsw)," https://www.pinecone.io/learn/series/faiss/hnsw/.

[7] "Openai gpt-3.5 turbo model," https://platform.openai.com/docs/models/gpt-3-5-turbo, accessed: 2024-05-07.

[8] "Google gemini api documentation," https://ai.google.dev/gemini-api/docs/get-started/python, accessed: 2024-05-07.

[9] "Meta llama 3-8b model on hugging face," https://huggingface.co/meta-llama/Meta-Llama-3-8B, accessed: 2024-05-07.

[10] “Cohere command-r documentation,” https://docs.cohere.com/docs/command-r, accessed: 2024-05-07.
[11] “Google flan-t5 base model on hugging face,” https://huggingface.co/google/flan-t5-base, accessed: 2024-05-07.