

# Semester 7: Project Elective Report

## Topic: Information Retrieval

**Team:** 1. Aryan Bhatt(IMT2020020),  
2. Jainav Sanghvi(IMT2020098),  
3. Ujjwal Agarwal(IMT2020128)

**Mentor:** Prof Milind Gandhe and Vivek Yadav

### Overall Objective of the project:

#### 1. Introduction:

- It was not until recent years that RAG has seen a resurgence of interest, due to the development of large language models(LLMs). LLMs are capable of generating text that is much more fluent and coherent than previous methods.

#### Current LLMs suffer from several drawbacks:

1. **They are static** - LLMs are “frozen in time” and lack up-to-date information. Their world exists as a static snapshot of the world as it was within their training data. This is called **data freshness problem**. It is not feasible to update their gigantic training datasets very frequently. Hence, ChatGPT has constraints due to its limited knowledge base, sometimes resulting in **hallucinating/nonsensical answers** when asked about unfamiliar topics .
2. **They lack domain-specific knowledge** - LLMs are trained for generalized tasks, meaning they do not know your company’s private data.
3. **They function as “black boxes”** - it’s not easy to understand which sources an LLM was considering when they arrived at their conclusions.
4. **They are inefficient and costly to produce** - Few organizations have the financial and human resources to produce and deploy foundation models.

Everyone does not have enough machines to train/finetune large language models. And How can you still get similar accuracy with smaller models? Augmentation of small sized language models with external memory is one approach. This technique is called **Retrieval Augmented Generation**.

RAG can help to address this problem by providing the LLM with a set of retrieved text that can be used to guide the generation process.

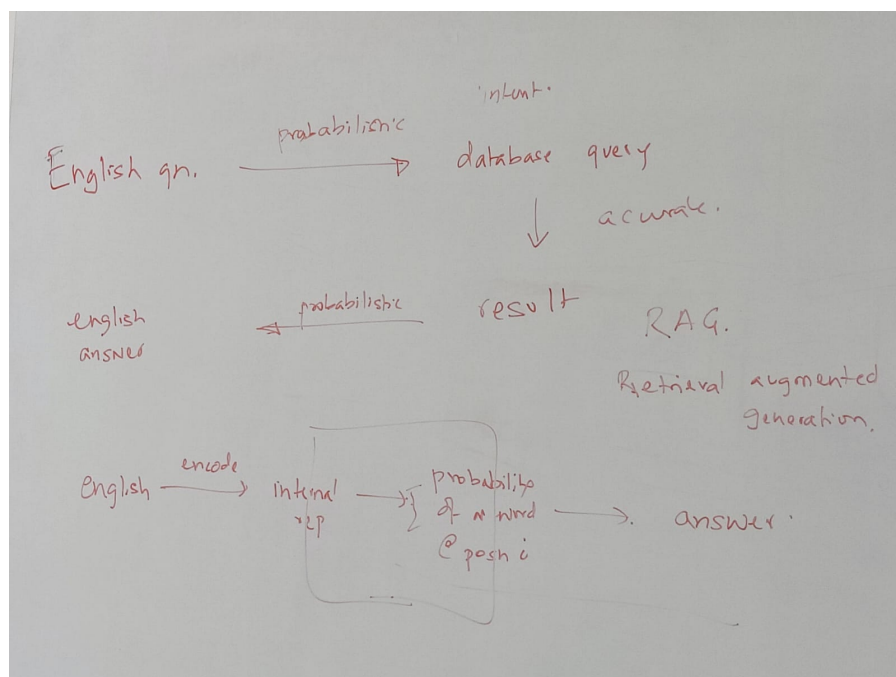
*Fine-Tuning modifies the internal parameters of the LLM to make it specialized, whereas RAG extends the model's capability by incorporating external knowledge dynamically during inference.*

### Why RAG?

- RAG can be applied to various natural language processing tasks, such as question answering, summarization, dialogue generation, and text completion.
- RAG can also help improve the domain adaptation and data freshness of an LLM, by enabling it to access up-to-date or specific information that is not available in its pre-trained data.
- **What are the benefits of RAG?** RAG reduces hallucination, facilitates fact-checking, enhances accuracy on domain specific tasks, and offers flexibility and cost-effectiveness for companies.
- **What are the challenges of RAG?** RAG depends on the quality of semantic search and existing data, introduces latency issues, and faces context length limitation.

## 2. Purpose of the Project (Problem Statement):

- The project aims to develop a specialized chatbot tailored for a specific institute or industry by utilizing a Retrieval Augmented Generation approach.
- The chatbot's primary function is to provide accurate and contextually relevant information sourced exclusively from the documents and knowledge base of the respective institute or industry.



### 3. Knowledge Integration:

- The chatbot's knowledge is derived directly from the official documents, policies, and information specific to the institute or industry, ensuring that responses are grounded in accurate and up-to-date information.
- By leveraging the institute or industry's documentation, the chatbot becomes a reliable source for users seeking information, reducing the risk of misinformation and enhancing the overall user experience.

### 4. Retrieval Augmented Generation Technique:

- The project employs a Retrieval Augmented Generation approach, combining the strengths of both retrieval-based and generative models.
- Retrieval mechanisms ensure that the chatbot's responses are rooted in pre-existing knowledge, while generation techniques enable flexibility in crafting responses, offering a balance between accuracy and natural language fluency.

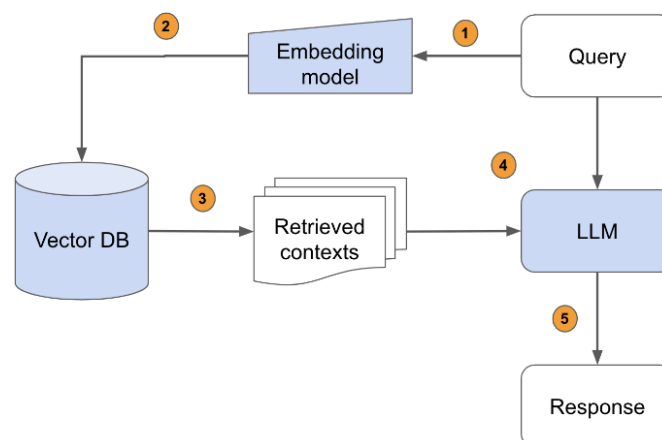
### 5. Mitigation of Information Hallucination:

- One of the key advantages of this approach is the reduction of information hallucination, a common challenge in chatbot systems. By relying on institute or industry-specific documents, the chatbot minimizes the risk of generating inaccurate or misleading responses.

### 6. Enhanced User Engagement and Satisfaction:

- Users interacting with the chatbot can expect precise and authoritative answers directly aligned with the institute or industry's expertise. This not only fosters trust but also enhances user satisfaction by delivering information that is both reliable and tailored to their specific needs.

By focusing on Retrieval Augmented Generation and utilizing the unique knowledge base of the institute or industry, the project aims to create a sophisticated chatbot that serves as a valuable resource for users while maintaining the integrity and accuracy of the information provided.



## Month Wise Progress:

### ❖ August

- Firstly researched the topic of Retrieval Augmented Generation and how chat bots actually work.
- Presentation of what RAG is: [RAG Basics PPT](#)
- Researched deeply the internal workings of chatbots like ChatGPT and BARD by Google and made comparisons between them in their working and capabilities.
- Gave a detailed presentation on how is Bing chat with Prometheus model different from ChatGPT: [Bing Chat Presentation Link](#)

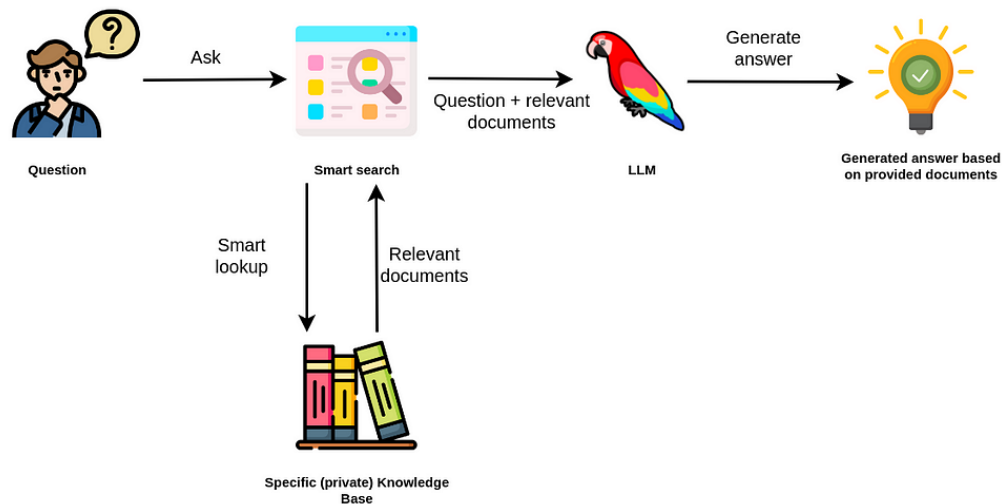
### ❖ September

- Presented a presentation about Prompt Engineering. Prepared the below presentation by reading a severe paper: [Prompt Engineering Presentation Link](#)
- Dwelled more in the concept of Retrieval Augmented Generation which totally aligns with the requirement of the project.
- Retrieval Augmented Generation (RAG) is an innovative approach that combines retrieval-based and generative techniques in natural language processing. It leverages pre-existing knowledge, often sourced from documents or databases, to provide contextually relevant and accurate responses, offering a balance between the specificity of the retrieved information and the flexibility of generative language models.
- Hence, by using the knowledge of only the documents and LLMs, we can make a chatbot that is specific to that institute.

RAG begins by employing a retrieval mechanism to search through a knowledge base or document repository. This mechanism identifies relevant passages or documents based on the input query.

Extracted information from the retrieval step is then integrated into the generative model's context. This integration ensures that the subsequent generation process is grounded in the retrieved knowledge, allowing the model to craft responses that align with the specific content found in the documents.

Following context integration, the generative model generates a response. This step allows for flexibility in language use, enabling the model to go beyond the retrieved information while still maintaining coherence and relevance. The result is a response that combines the specificity of retrieved facts with the fluency of generative language models.

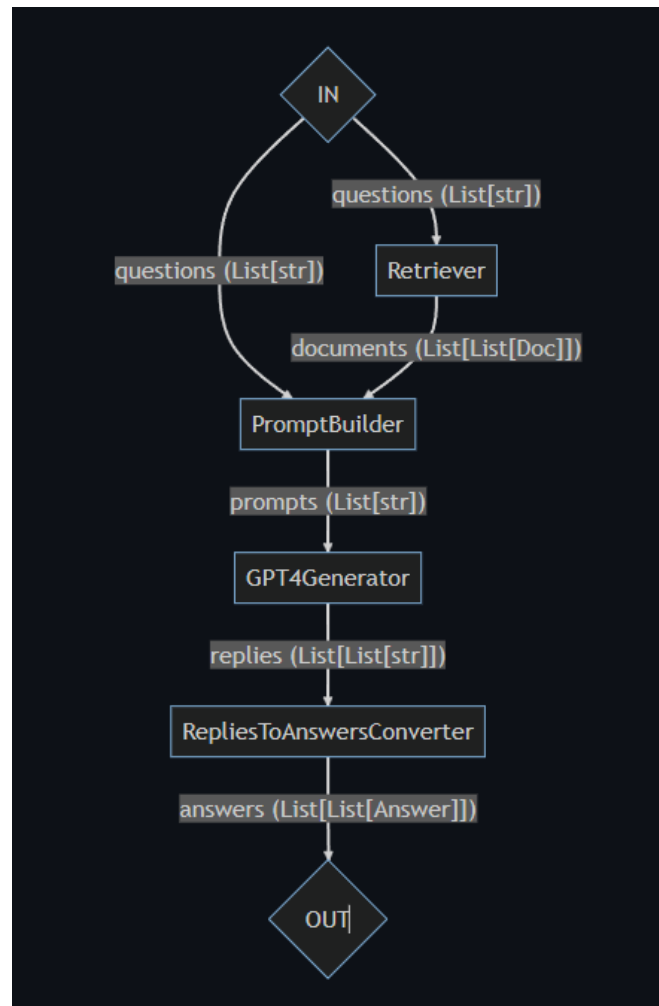


In the subsequent months of the project, focussed on how to do this RAG implementation and what frameworks can be used to implement this.

## ❖ October

- Encountered with **Haystack**, a python-based library which helps to build LLM-based pipelines and can help to implement RAG in the project. In a RAG pipeline, the query input is passed on to the retriever, powered by a LLM, the retriever retrieves the relevant documents based on the query and the context of the retrieved documents the LLM forms an output sentence which is accurate to the documents provided in the database.
- Haystack is an open-source Python library designed for building scalable and efficient question-answering systems using state-of-the-art transformer models. It streamlines the development of search and retrieval components
- Documentation: [HayStack Documentation](#)
- Haystack Presentation: [HayStack PPT](#)
- From providing databases and database searches (like ElasticSearch, Weaviate Search, etc) to providing retrievers, haystack supports all of them. It also supports Open-source LLMs of Hugging Face (falcon-7b) in its interface.

## RAG Pipeline in Haystack 2.0



## Components of HayStack

### PipeLine

- A "pipeline" typically refers to a sequence of processing steps that are applied to a user's input to generate a response. These pipelines are crucial for transforming raw input (such as a user's query) into a structured format that can be used to retrieve relevant information and generate a meaningful response.
- It consist of many different components like:
  - **Reader:** The first step in the pipeline is often a reader, which is responsible for extracting information from the user's input. This

could involve tasks like text extraction, document retrieval, or any other form of information retrieval.

- **Document Store:** In some cases, the preprocessed data is stored in a document store, a database or index that facilitates efficient retrieval of relevant information. **Retriever:** The retriever component is responsible for searching the document store to identify the most relevant documents or passages related to the user's query.
- **Prompt Node:** PromptNode brings you the power of large language models. It's an easy-to-use, customizable node that you can run on its own or in your pipelines for various NLP tasks. Through this only, we get embeddings of the query of the user. We combine the context of the retrieved documents and the context of the query and output it as the final answer in sentences on the chatbot. Can support many open source LLMs like falcon-7b, google-flat-base y5, mistral 7b, etc.

## Experimented on various LLMs

We experimented on various LLMs. We found that base-models like google-flat-base-t5 and all other models like this are made to predict the next word. They are not meant for generation of text like OPEN AI models (did not use OPEN AI models because we do not want our private institute / industry data to some third party company). Hence, we needed some LLMs that can generate sentences. So, we used falcon-7b and mistral 7b for this and they generated some good response outputs. But still we were facing problems of hallucinations which were discussed afterwards.

Here are the code snippets of the components of the pipeline:

```
from haystack import Pipeline
from haystack.document_stores import WeaviateDocumentStore
from haystack.nodes import EmbeddingRetriever, PromptNode

document_store = WeaviateDocumentStore()
retriever = EmbeddingRetriever(document_store=document_store,
                               embedding_model="sentence-transformers/multi-qa-mpnet-base-dot-v1")
prompt_node = PromptNode(model_name_or_path="gpt-4",
                          default_prompt_template="deepset/question-answering",
                          api_key='YOUR_OPENAI_KEY')

p = Pipeline()
p.add_node(component=retriever, name="EmbeddingRetriever", inputs=["Query"])
p.add_node(component=prompt_node, name="QAPromptNode", inputs=["EmbeddingRetriever"])
res = p.run(query="What did Einstein work on?")
```

Here it depicts how to make a standard RAG pipeline using prompt node and retriever using Weaviate as a document store.

```

query_pipeline = Pipeline()
query_pipeline.add_node(component=retriever, name="Retriever", inputs=["Query"])
query_pipeline.add_node(component=prompt_node, name="PromptNode", inputs=["Retriever"])

from haystack.utils import print_answers

response = query_pipeline.run("What is Taj Mahal", params={"Retriever" : {"top_k": 1}})
print_answers(response, details='all')

```

In this, a Pipeline object (predefined by haystack) is initiated and then we add nodes to the pipeline. There are two nodes: Retriever and Prompt Node. Retriever basically retrieves the relevant documents based on the query of the user. Prompt Node is basically a LLM only that combines the context of the documents retrieved and the query to make a combined output at the end.

## Vector Database

- The documents are not stored as simple PDFs. It is due to the fact that we need the relevant documents. Hence, the documents that match the context of the query should be retrieved (not only because it has the same words as the query).
- For this we apply similarity (like cosine similarity) to the documents and the query to implement the document search.
- Hence, the documents are also converted into embeddings and stored in a vector database: have used Weaviate for this. Link: <https://weaviate.io/>
- Update Embeddings were implemented in them to further refine the embeddings.

## Normal Database vs Vector Database

- A normal database is a type of database that stores data in a structured or semi-structured way, such as tables, documents, or graphs. A normal database can handle various types of data, such as text, numbers, dates, and binary. A normal database is good for storing and querying data based on exact matches, but it may not be efficient for finding similar or related data.
- A vector database is a type of database that stores data as vectors, which are numerical representations of data objects, also known as vector embeddings. A vector database can handle high-dimensional data, such as images, audio, video, and natural language. A vector database is good for storing and querying data based on similarity or distance measures, but it may not be efficient for finding complex relationships or patterns in the data.



## Collab Demo

In recent weeks have tried to implement the RAG model in Haystack. The link for the collab is : [🔗 Weaviate.ipynb](#)

- First we installed haystack from GitHub.
- Then we initiated a Weaviate database node so that we could perform Weaviate search on the documents. For this, we initialized a Weaviate node on their website and accessed it through the API key.
- For this demo, a document of seven wonders of the world was taken which contains the information regarding the seven wonders of the world.
- So the documents were stored in the document store and converted to embeddings.
- Then the retriever as the Embedding Retriever was initialized which will search the documents as per the closest embeddings.
- Prompt Node was also initialized with a LLM of falcon-7b from hugging face.


```
# from hugging face api
prompt_node = PromptNode(model_name_or_path = "tiiuae/falcon-7b-instruct",
                          default_prompt_template = prompt_template, use_gpu=True, api_key = "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")
```

- A Prompt template was also initialized which instructed the prompt node on how to make the output sentence finally.

```
prompt_template = PromptTemplate(
    prompt="""
    Answer the question truthfully based solely on the given documents. If the documents do not contain the answer to the question,
    say that answering is not possible given the available information. Your answer should be no longer than 50 words.
    Documents:{join(documents)}
    Question:{query}
    Answer:
    """,
    output_parser=AnswerParser(),
)
```

- Finally, the output sentences were produced when asked with some query related to the seven wonders of the world.

## ❖ October - November

- Conducted in-depth research on Knowledge and Vector Databases, specifically exploring Weaviate. Delved into knowledge graphs utilizing Weaviate technology.
-  Knowledge graphs with Weaviate
- Uncovered a new tool, Verba, during our investigation. Developed a comprehensive presentation, titled "Verba Presentation," outlining the intricacies of Verba. [Verba Presentation](#)
- Understood and Presented the Verba Architecture and its functionality, supplemented by a live demonstration. For additional details, please refer to the Verba presentation.
- Successfully installed Verba locally on our laptop, necessitating code modifications for optimal performance. Tested Verba with various documents using the llama 7b model, achieving satisfactory results. However, in pursuit of improved output, we experimented with the llama 13b and llama 70b models. Unfortunately, we encountered a disk space limitation on the server, preventing the download of all parameter files for the llama models.
- The challenges faced were promptly communicated to Prof. Milind and Prof. Vivek for resolution.

### **Researched about different search mechanisms:**

A more detailed presentation on these database search techniques is present in the: [HayStack PPT](#)

**Elastic Search:** Elasticsearch is an open-source, distributed search and analytics engine. It is designed to store, search, and analyze large volumes of data quickly and in near real-time. Elasticsearch is designed to scale horizontally, allowing it to handle large amounts of data by distributing it across multiple nodes in a cluster. This architecture enables Elasticsearch to deliver high performance and availability. One of Elasticsearch's primary use cases is full-text search. It supports powerful search capabilities, including fuzzy matching, wildcards, and complex queries, making it suitable for applications requiring advanced search functionality.

**Open Search:** OpenSearch is a fork of Elasticsearch, another popular search and analytics engine. The project was created as a community-driven alternative

after concerns arose about the direction of Elasticsearch's licensing and development.

OpenSearch is designed to be distributed, allowing it to scale horizontally across multiple nodes to handle large amounts of data and traffic. It supports features like sharding and replication for data distribution and fault tolerance. OpenSearch can be used for full-text search, log analytics, and other data analysis tasks. It supports complex queries, aggregations, and various types of data visualization. But it is proved that in many aspects, elastic search is better than open search.

**Weaviate Search:** Weaviate is an open source vector database.

Weaviate is a database engine that can store data in JSON format and perform neural search on the data by vectorizing it and building approximate nearest neighbor indices. This allows for efficient and accurate retrieval of relevant information based on similarity of vectors.

Weaviate is designed to provide semantic search capabilities, allowing users to search for and retrieve information based on the meaning or context of the data rather than just keywords. Weaviate focuses on semantic search, which means it understands the context and relationships between entities in the data. This enables more contextually relevant search results compared to traditional keyword-based search engines. Weaviate uses vector embeddings to represent data entities in a multidimensional space. This representation allows for efficient similarity searches, where similar entities are located in proximity to each other in the vector space.

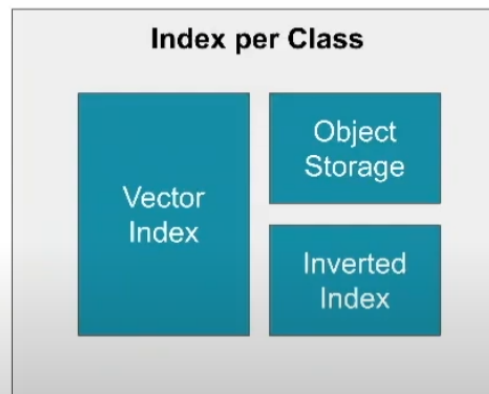
Vector database: Weaviate is a fantastic tool for retrieving the information you need, quickly and accurately. It does this by being an amazing vector database.

As we have to store the context of the documents and search them based on the context only. We convert the documents into vectors and store it in the weaviate datastore. Then we apply an embedding retrieval of Haystack and apply search on these embeddings of documents to search the most relevant (like top 2, 3) documents.



## How Data is stored in Weaviate

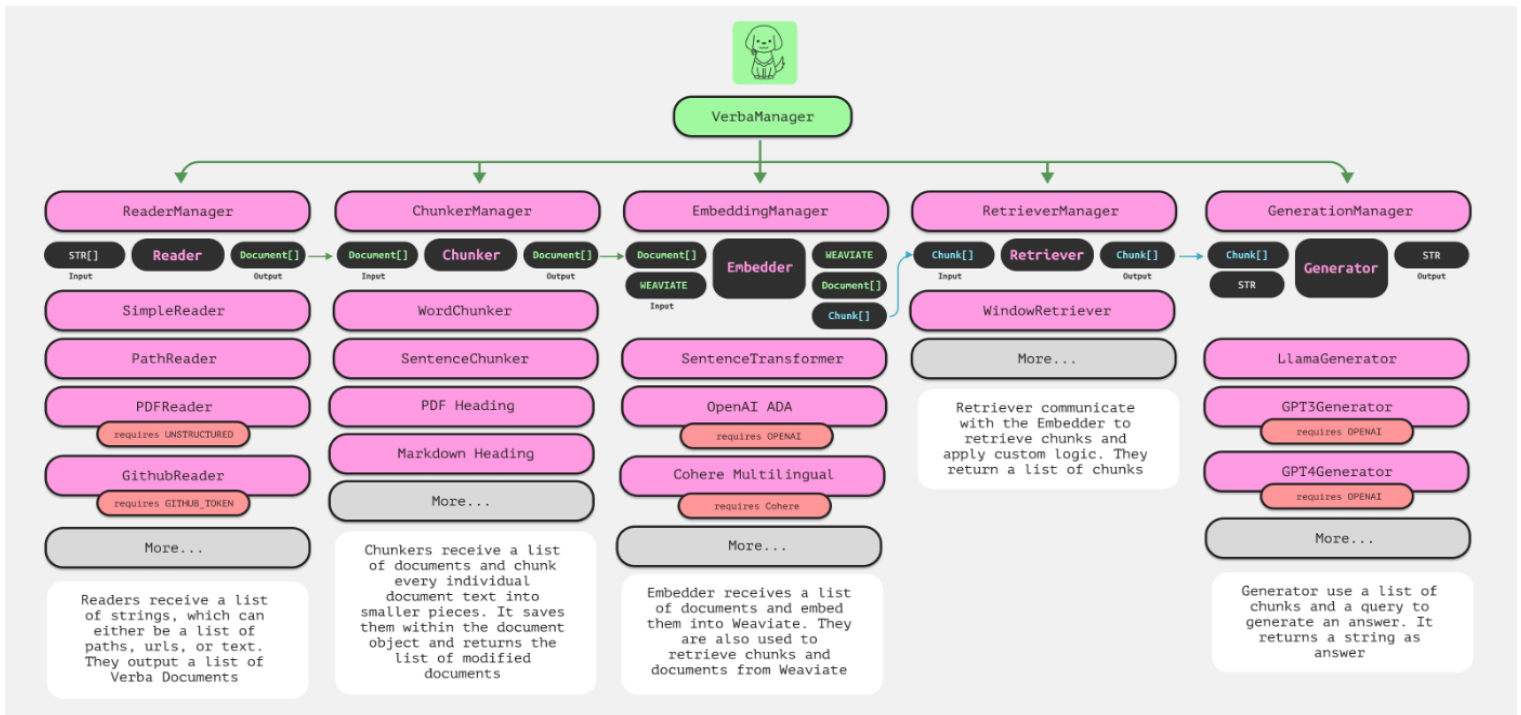
- A vector search returns the whole object (from disk), not just an ID
- You can combine structured filters (e.g. "accountBalance <= 2000") with vector search
- Any mutation is immediately persisted and safe against crashes



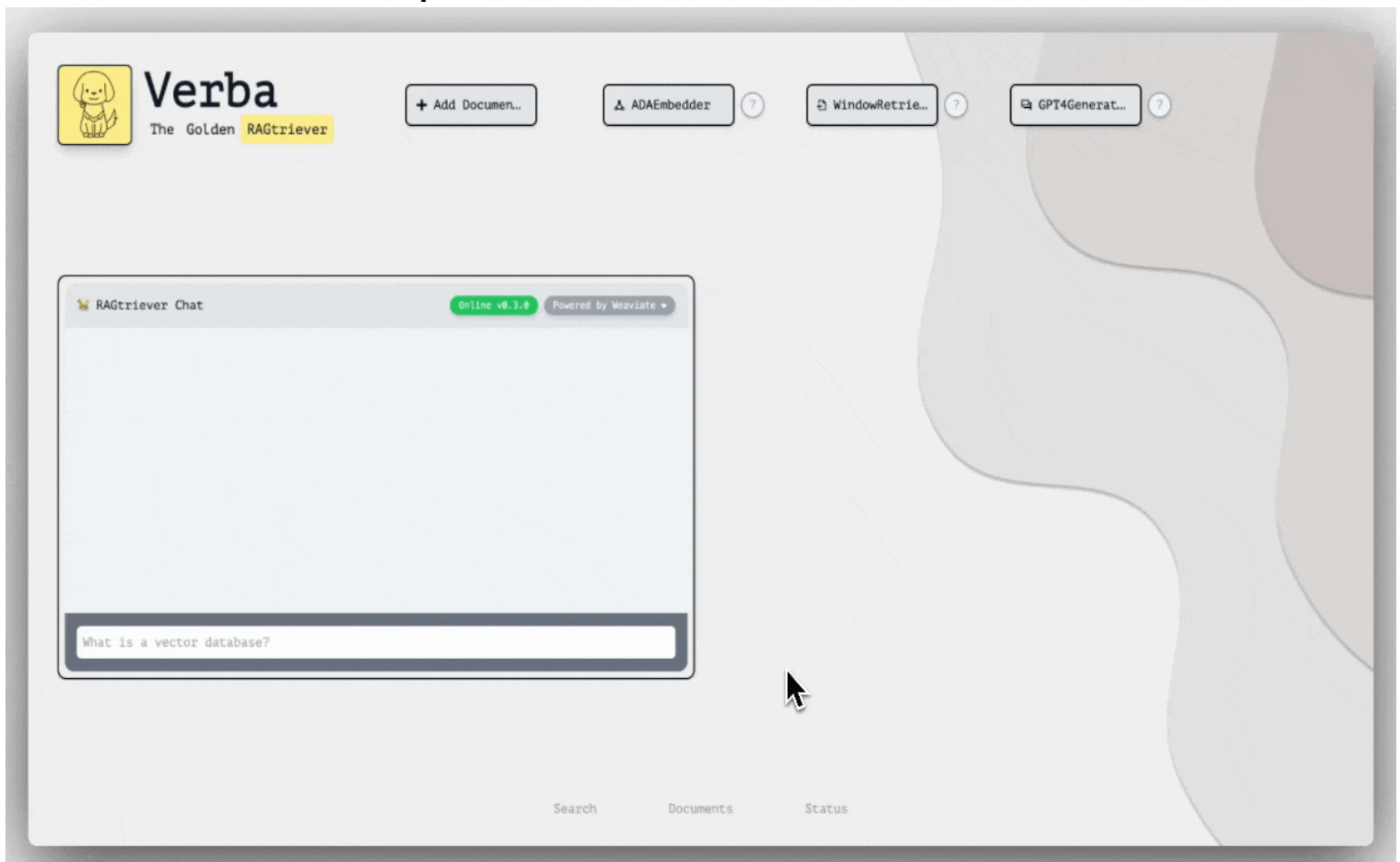
## What is Verba?

- ❖ It is an open-source application designed to offer an end-to-end, streamlined, and user-friendly interface for Retrieval-Augmented Generation (RAG).
- ❖ Empowers with Weaviate's context-aware database and the analytical power of Large Language Models (LLMs).
- ❖ Intuitive chat interface refines search results based on ongoing conversation context for accurate and relevant information.
- ❖ Supports tools like Weaviate, OpenAI, Cohere, and HuggingFace (Llama2), Unstructured.
- ❖ For more details about Verba, please refer to the presentation we made regarding Verba


## Verba Architecture



## Verba Demo Examples:




## How does Verba deal with questions out of its knowledge base?(Hallucinations)

 RAGtriever Chat

Online v0.3.1

Powered by Weaviate

where is IIITB located?

 Copy

The provided context does not contain information about the location of IIITB (International Institute of Information Technology Bangalore). The context is related to various accommodations and their details in Brooklyn and Manhattan, New York City. If you are looking for information about IIITB, please provide relevant context or details about the institute.

What is a vector database?

## **Strategic Directions to take:**

- ❖ Use of Lang Chain for Implementation. Use open source LLMs like Mistral and Falcon. Can make a proper web app using it.
- ❖ Work on Verba as it has the potential to solve our use case.

## **Key Learnings from the Project:**

- ❖ Chat bots are not always to be trusted as they come up with the problem of hallucination. We learned that how chatbots like Bing AI, ChatGPT and google BARD works under the hood.
- ❖ Most importantly, we learned the concept of Retrieval Augmented Generation (RAG) in which we aim to minimize hallucination by giving answer responses only on the basis of the documents stored in the database. If the document does not contain the information to answer the query then the chat bot responds with that it doesn't know. Researched extensively about RAGs and read many articles and research papers about it online.
- ❖ Also learned about the concepts of vector databases. They are different from traditional SQL and NOSQL databases. They are mostly used for searching and querying. They make querying of documents faster (instead of using BM25, TF-IDF, etc which ranks the documents on the basis of word frequency). Moreover, as they convert the documents to embeddings, they also store the context. Hence, vector search also enables context based search.
- ❖ Looked at various implementations of RAG and how to implement it according to our needs. Researched through many open source implementations of RAGs to get a grasp of how to implement RAG systems on a set of documents.

# Haystack

## Problems

- The problem of hallucinations still persists for irrelevant query. Even though the prompt template clearly specifies that if the document does not contain any information regarding the query then it should output that the document does not contain the information.

## Solution

- Possibly the hallucinations are due to the fact the LLM is using its own general knowledge to answer the query. We also tried with a different LLM (google flan-t5-base) but this LLM is used for word generation. Hence, it will produce only a single word. In that, no hallucination was observed.
- Another approach(yet to be implemented) is to use **LangChain** as a pipeline library instead of Haystack. It also provides vector databases and document retrievers and can be arranged in a pipeline. It also integrates well with hugging face open-source LLM models.

# Verba

## Problems

- We tested Verba with various documents using the llama 7b model, achieving satisfactory results. However, in pursuit of improved output, we experimented with the llama 13b and llama 70b models. Unfortunately, we encountered a disk space limitation on the server, preventing the download of all parameter files for the llama models.
- We also attempted to use the GPT model for output generation, but since we didn't obtain a license for GPT, we encountered the error: "Something went wrong! Object of type 'NoneType' has no len()." [Issue](#)

## Solutions

- If we can obtain additional storage, we could try using better models like Llama 70b. Additionally, obtaining a license for the GPT model would allow us to achieve better results with no hallucination.



## References

- <https://docs.haystack.deepset.ai/docs>
- <https://huggingface.co/spaces/deepset/retrieval-augmentation-svb>
- <https://github.com/weaviate/Verba.git>
- <https://verba.weaviate.io/>
- <https://unstructured-io.github.io/unstructured/>