**What is Retrieval Augmented Generation (RAG)?**

Explore Retrieval Augmented Generation (RAG) RAG: Integrating LLMs with data search for nuanced AI responses. Understand its applications and impact.

Updated Jan 30, 2024 · 8 min read

**What is RAG?**

RAG, or Retrieval Augmented Generation, is a technique that combines the capabilities of a pre-trained large language model with an external data source. This approach combines the generative power of LLMs like GPT-3 or GPT-4 with the precision of specialized data search mechanisms, resulting in a system that can offer nuanced responses.

This article explores retrieval augmented generation in more detail, giving some practical examples and applications, as well as some resources to help you learn more about LLMs. To get started, check out our course on [**mastering LLM concepts**](https://www.datacamp.com/courses/large-language-models-llms-concepts). You can also view our code-along below onRetrieval Augmented Generation with PineCone.

**Why Use RAG to Improve LLMs? An Example**

To better demonstrate what RAG is and how the technique works, let’s consider a scenario that many businesses today face.

Imagine you are an executive for an electronics company that sells devices like smartphones and laptops. You want to create a customer support chatbot for your company to answer user queries related to product specifications, troubleshooting, warranty information, and more.

You’d like to use the capabilities of LLMs like GPT-3 or [**GPT-4**](https://www.datacamp.com/blog/what-we-know-gpt4) to power your chatbot.

However, large language models have the following limitations, leading to an inefficient customer experience:

**Lack of specific information**

Language models are limited to providing generic answers based on their training data. If users were to ask questions specific to the software you sell, or if they have queries on how to perform in-depth troubleshooting, a traditional LLM may not be able to provide accurate answers.

This is because they haven’t been trained on data specific to your organization. Furthermore, the training data of these models have a cutoff date, limiting their ability to provide up-to-date responses.

**Hallucinations**

LLMs can “hallucinate,” which means that they tend to confidently generate false responses based on imagined facts. These algorithms can also provide responses that are off-topic if they don’t have an accurate answer to the user’s query, leading to a bad customer experience.

**Generic responses**

Language models often provide generic responses that aren’t tailored to specific contexts. This can be a major drawback in a customer support scenario since individual user preferences are usually required to facilitate a personalized customer experience.

RAG effectively bridges these gaps by providing you with a way to integrate the general knowledge base of LLMs with the ability to access specific information, such as the data present in your product database and user manuals. This methodology allows for highly accurate and reliable responses that are tailored to your organization’s needs.

**How Does RAG Work?**

Now that you understand what RAG is, let’s look at the steps involved in setting up this framework:

**Step 1: Data collection**

You must first gather all the data that is needed for your application. In the case of a customer support chatbot for an electronics company, this can include user manuals, a product database, and a list of FAQs.

**Step 2: Data chunking**

Data chunking is the process of breaking your data down into smaller, more manageable pieces. For instance, if you have a lengthy 100-page user manual, you might break it down into different sections, each potentially answering different customer questions.

This way, each chunk of data is focused on a specific topic. When a piece of information is retrieved from the source dataset, it is more likely to be directly applicable to the user’s query, since we avoid including irrelevant information from entire documents.

This also improves efficiency, since the system can quickly obtain the most relevant pieces of information instead of processing entire documents.

**Step 3: Document embeddings**

Now that the source data has been broken down into smaller parts, it needs to be converted into a vector representation. This involves transforming text data into embeddings, which are numeric representations that capture the semantic meaning behind text.

In simple words, document embeddings allow the system to understand user queries and match them with relevant information in the source dataset based on the meaning of the text, instead of a simple word-to-word comparison. This method ensures that the responses are relevant and aligned with the user’s query.

If you’d like to learn more about how text data is converted into vector representations, we recommend exploring our tutorial on [**text embeddings with the OpenAI API**](https://www.datacamp.com/tutorial/introduction-to-text-embeddings-with-the-open-ai-api).

**Step 4: Handling user queries**

When a user query enters the system, it must also be converted into an embedding or vector representation. The same model must be used for both the document and query embedding to ensure uniformity between the two.

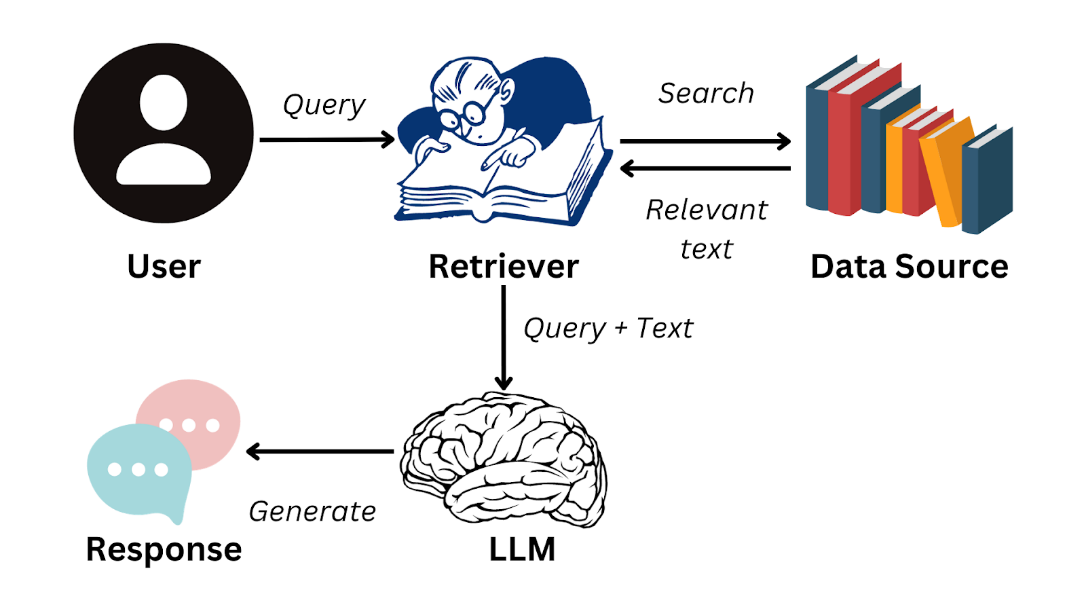
Once the query is converted into an embedding, the system compares the query embedding with the document embeddings. It identifies and retrieves chunks whose embeddings are most similar to the query embedding, using measures such as cosine similarity and Euclidean distance.

These chunks are considered to be the most relevant to the user’s query.

**Step 5: Generating responses with an LLM**

The retrieved text chunks, along with the initial user query, are fed into a language model. The algorithm will use this information to generate a coherent response to the user’s questions through a chat interface.

Here is a simplified flowchart summarizing how RAG works:

**

*Image by author*

To seamlessly accomplish the steps required to generate responses with LLMs, you can use a data framework like LlamaIndex.

This solution allows you to develop your own LLM applications by efficiently managing the flow of information from external data sources to language models like GPT-3. To learn more about this framework and how you can use it to build LLM-based applications, read our [**tutorial on LlamaIndex**](https://www.datacamp.com/tutorial/llama-index-adding-personal-data-to-llms).

**Practical Applications of RAG**

We now know that RAG allows LLMs to form coherent responses based on information outside of their training data. A system like this has a variety of business use cases that will improve organizational efficiency and user experience. Apart from the customer chatbot example we saw earlier in the article, here are some practical applications of RAG:

**Text summarization**



*Image by DALLE-3*

RAG can use content from external sources to produce accurate summaries, resulting in considerable time savings. For instance, managers and high-level executives are busy people who don’t have the time to sift through extensive reports.

With an RAG-powered application, they can quickly tap into the most critical findings from text data and make decisions more efficiently instead of having to read through lengthy documents.

**Personalized recommendations**

RAG systems can be used to analyze customer data, such as past purchases and reviews, to generate product recommendations. This will increase the user’s overall experience and ultimately generate more revenue for the organization.

For example, RAG applications can be used to recommend better movies on streaming platforms based on the user’s viewing history and ratings. They can also be used to analyze written reviews on e-commerce platforms.

Since LLMs excel at understanding the semantics behind text data, RAG systems can provide users with personalized suggestions that are more nuanced than those of a traditional recommendation system.

**Business intelligence**

Organizations typically make business decisions by keeping an eye on competitor behavior and analyzing market trends. This is done by meticulously analyzing data that is present in business reports, financial statements, and market research documents.

With an RAG application, organizations no longer have to manually analyze and identify trends in these documents. Instead, an LLM can be employed to efficiently derive meaningful insight and improve the market research process.

**Challenges and Best Practices of Implementing RAG Systems**

While RAG applications allow us to bridge the gap between information retrieval and natural language processing, their implementation poses a few unique challenges. In this section, we will look into the complexities faced when building RAG applications and discuss how they can be mitigated.

**Integration complexity**

It can be difficult to integrate a retrieval system with an LLM. This complexity increases when there are multiple sources of external data in varying formats. Data that is fed into an RAG system must be consistent, and the embeddings generated need to be uniform across all data sources.

To overcome this challenge, separate modules can be designed to handle different data sources independently. The data within each module can then be preprocessed for uniformity, and a standardized model can be used to ensure that the embeddings have a consistent format.

**Scalability**

As the amount of data increases, it gets more challenging to maintain the efficiency of the RAG system. Many complex operations need to be performed - such as generating embeddings, comparing the meaning between different pieces of text, and retrieving data in real-time.

These tasks are computationally intensive and can slow down the system as the size of the source data increases.

To address this challenge, you can distribute computational load across different servers and invest in robust hardware infrastructure. To improve response time, it might also be beneficial to cache queries that are frequently asked.

The implementation of vector databases can also mitigate the scalability challenge in RAG systems. These databases allow you to handle embeddings easily, and can quickly retrieve vectors that are most closely aligned with each query.

If you’d like to learn more about the implementation of vector databases in an RAG application, you can watch our live code-along session, titled [**Retrieval Augmented Generation with GPT and Milvus**](https://www.datacamp.com/code-along/retrieval-augmented-generation-with-gpt-and-milvus). This tutorial offers a step-by-step guide to combining Milvus, an open-source vector database, with GPT models.

**Data quality**

The effectiveness of an RAG system depends heavily on the quality of data being fed into it. If the source content accessed by the application is poor, the responses generated will be inaccurate.

Organizations must invest in a diligent content curation and fine-tuning process. It is necessary to refine data sources to enhance their quality. For commercial applications, it can be beneficial to involve a subject matter expert to review and fill in any information gaps before using the dataset in an RAG system.

**Final Thoughts**

RAG is currently the best-known technique to leverage the language capabilities of LLMs alongside a specialized database. These systems address some of the most pressing challenges encountered when working with language models, and present an innovative solution in the field of natural language processing.

However, like any other technology, RAG applications have their limitations - particularly their reliance on the quality of input data. To get the most out of RAG systems, it is crucial to include human oversight in the process.

The meticulous curation of data sources, along with expert knowledge, is imperative to ensure the reliability of these solutions.

If you’d like to dive deeper into the world of RAG and understand how it can be used to build effective AI applications, you can watch our live training on [**building AI applications with LangChain**](https://www.datacamp.com/code-along/building-ai-applications-with-langchain-and-gpt). This tutorial will give you hands-on experience with LangChain, a library designed to enable the implementation of RAG systems in real-world scenarios.