LLM Retrieval-Augmented Generation (RAG) with OpenVINO™ and LangChain

1. Executive Summary

Retrieval-Augmented Generation (RAG) is one of the most efficient and inexpensive ways for companies to create their own AI applications around Large Language Models (LLMs). It allows LLMs to augment their knowledge with an additional information source specific to a certain domain. RAG extends the capability of the LLM and improves response quality without needing to go through the time-consuming process of fine-tuning.

Companies that want to deploy an AI application (such as a support chatbot) can use OpenVINO™ and LangChain to implement an efficient RAG pipeline. OpenVINO™ provides the following benefits:

- Best-in-class performance for served LLMs with Intel® Core™ and Intel® Xeon® processors
- Support for a variety of LLM architectures from a wide range of frameworks
- Weight compression and optimization with NNCF
- Pre-converted and optimized models available

This white paper gives more information about RAG pipelines, why they are useful, and how they work. It shows code examples for each stage of the pipeline and includes links to end-to-end examples showing how to deploy chatbot applications locally or using OpenVINO™ Model Server.

2. Introduction to Retrieval-Augmented Generation

Large Language Models (LLMs) are deep learning neural networks that have been trained on large text datasets scraped from the Internet and other sources. They are being used in an increasing number of AI-based products and services, such as chat support agents, virtual assistants, code generators, and much more. However, there are two major drawbacks with LLMs:

1. They only have knowledge of topics up to a certain date. For example, ChatGPT-4 does not have knowledge of events from April 2023 onward.
2. The models "hallucinate" in their response, and confidently give wrong answers on topics they don't know about.
3. Publicly available models have no access to private data and are unable to generate any content about proprietary information.

Businesses seeking to deploy their own LLM can overcome these limitations by fine-tuning models on a custom dataset. However, the process of curating a dataset for fine-tuning takes significant effort and time.

Retrieval-Augmented Generation (RAG) is a method for augmenting a LLM's knowledge with specific data beyond the general dataset it was trained on. It increases the quality and accuracy of LLMs without needing to fine-tune a custom model. With RAG, the LLM is provided with a set of documents to retrieve information from, and then it answers user queries with the information from the documents. This way, the model has immediate access to the necessary information and can use it to generate an accurate response.

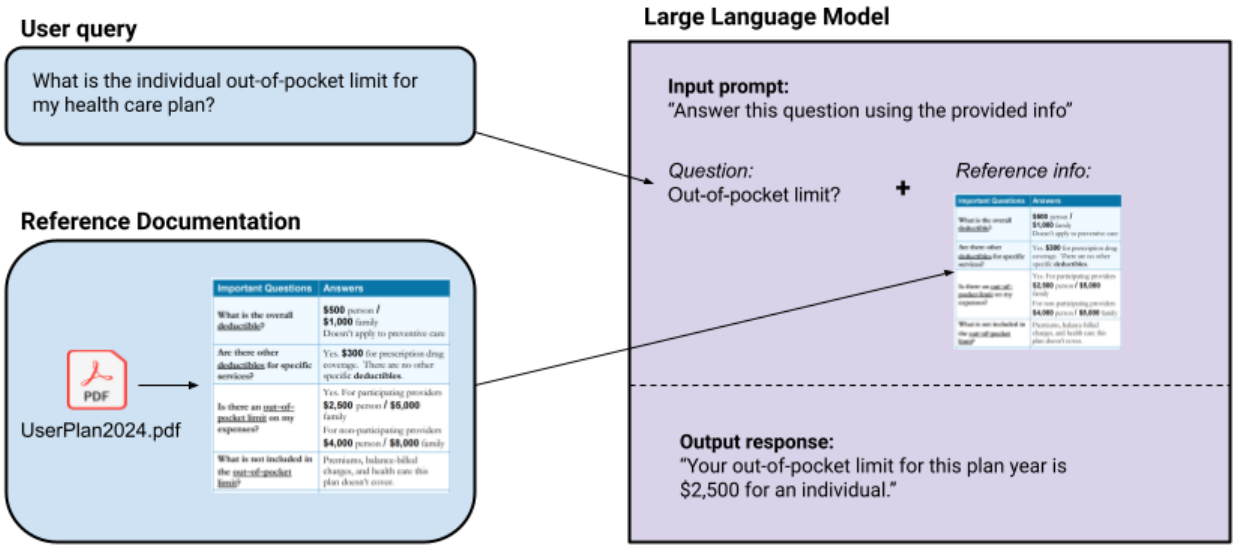

Figure 1. A basic RAG example where reference information is used to help a user's question.

There are several benefits to using RAG:

1. RAG increases the accuracy of LLM responses, because the LLM can directly reference the set of information provided rather than relying on its general knowledge.
2. It significantly reduces the likelihood of hallucination and incorrect responses. If the provided documentation does not have the information the user is looking for, the LLM can simply say it doesn't know the answer to the user's query.
3. The LLM's knowledge source can be updated in real time so it can stay current with changes in information.
4. RAG responses are more transparent because they can include references to the source of the information.

RAG allows companies to develop AI-enabled chatbots with specific knowledge of their product or service without needing to go through the costly and time-consuming process of fine tuning. For example, a health insurance company can provide an LLM with information about its health care plans to give it knowledge of their policies and rates. Then, when a customer asks about premiums and coverage for a certain plan, the RAG-empowered chat agent can respond with detailed and accurate information. If portions of the plan change, the company can immediately upload the changed documentation so the chatbot stays up-to-date with current knowledge.

This white paper explains how a RAG pipeline works, shows how to set one up with OpenVINO™ and LangChain, and links to comprehensive end-to-end examples for deploying pipelines locally or on a server.

3. Explanation of RAG Pipeline

3.1. RAG Pipeline

A typical RAG pipeline consists of several stages and may use more than one deep learning model in the question-answering process. An example full pipeline is shown below.

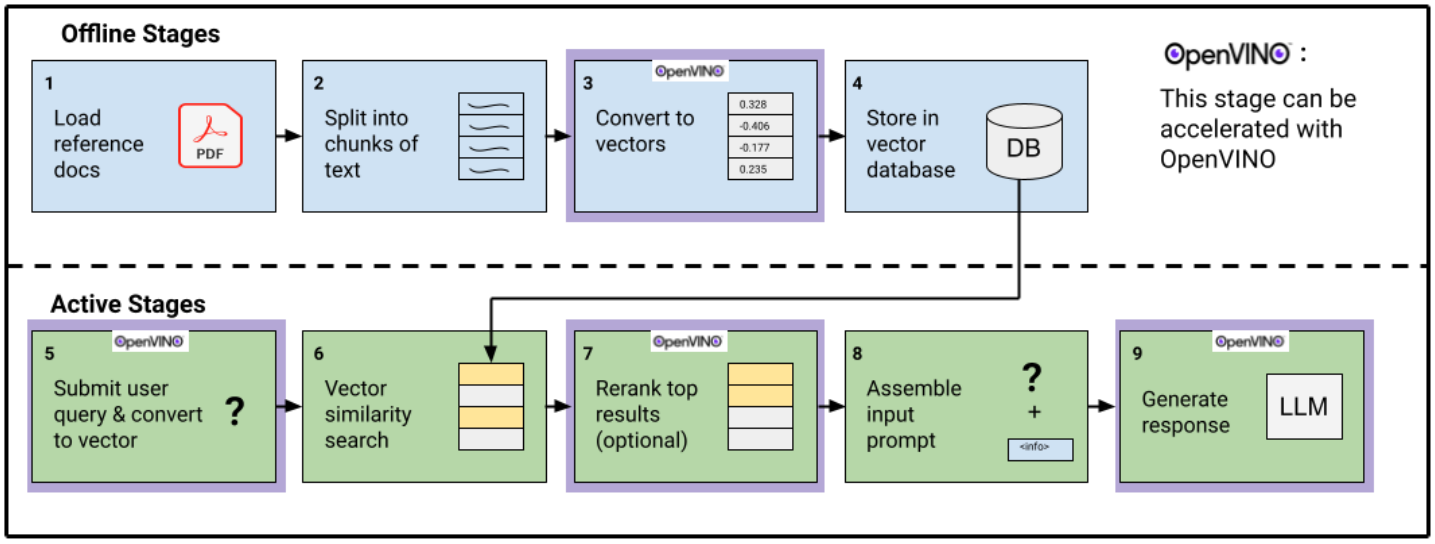

Figure 2. The RAG pipeline consists of preparation stages that occur offline (before deployment) and active stages that occur when the user is interacting with the application.

The first four stages occur offline, before the model is deployed:

1. The source documents are loaded using an unstructured loader (which supports .txt, .pdf, .html, etc).
2. The documents are split into chunks of text, making them easier to parse. The chunks of text have to be short enough to fit in the LLM's context window and must be small enough to be accurately representable by a single embedding.
3. These text chunks are converted to vectors that numerically represent the information in the text using an embedding model.
4. The vectors are stored in an indexed database which can easily be searched through.

When the application is active and a user submits an input prompt:

5. The user's text input prompt is converted to a vector using the same embedding model.
6. The query vectors are mathematically compared to the document vectors to determine which portions of documentation are most relevant to the query, returning the "top k" matches.
7. (Optional) A cross-encoder based rerank model is used to sort the top k document matches. Rerank models use transformer networks to determine the similarity of the

query to the document chunks, so they may provide a better order of relevance to the query.
8. The relevant chunks of documentation are added as context to the original user input and submitted to the LLM.
9. The LLM returns an answer based on the user input and the context of the documentation it was provided.

By following this process, the RAG pipeline allows the LLM to provide an answer based on the relevant documentation, which will be more accurate and less prone to hallucinations.

Three different deep learning models are used in the RAG pipeline: an embedding model, a rerank model, and a large language model. Each of these can be optimized and compressed using OpenVINO™ for better system performance. The sections below provide further explanation of each model.

3.2. Text Embedding Models

A fundamental part of RAG operation is text embedding. This is the process of converting a sequence of words (text) into a sequence of vectors (numbers) that represent the information contained in the words. Embeddings make it so deep learning models and other algorithms can numerically calculate the relationships between pieces of text, allowing them to perform tasks such as clustering or retrieval. In the case of RAG, embeddings are used to compare the user input to the stored documents so the pipeline can retrieve the pieces of text that are similar to the user query. It occurs at the “Embedding” stages in the pipeline diagram shown in Section 3.1.

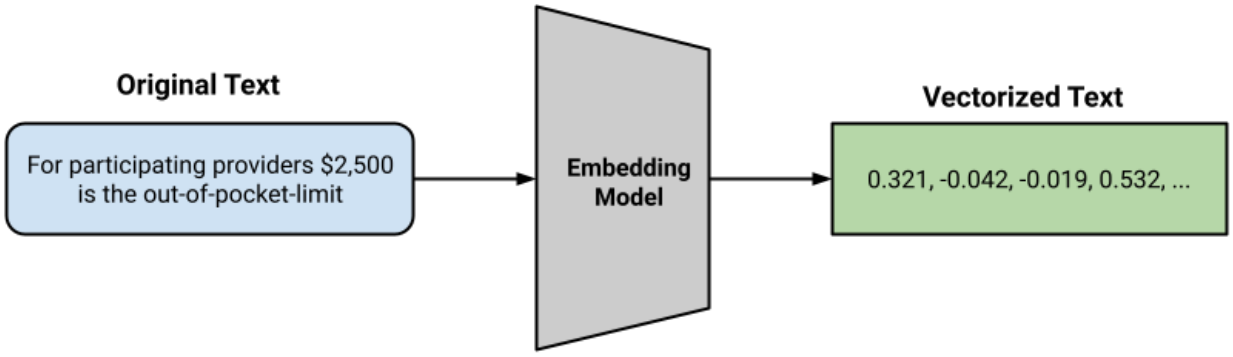

Figure 3. Embedding models convert text into high-dimensional numerical vectors.

Converting a series of words into a series of vectors that contain the semantic meaning of the words is not a straightforward task. It requires the use of embedding models, which are deep learning models trained to efficiently encode words into high-dimensional vectors. A list of popular embedding models can be found at the Massive Text Embedding Benchmark Leaderboard.