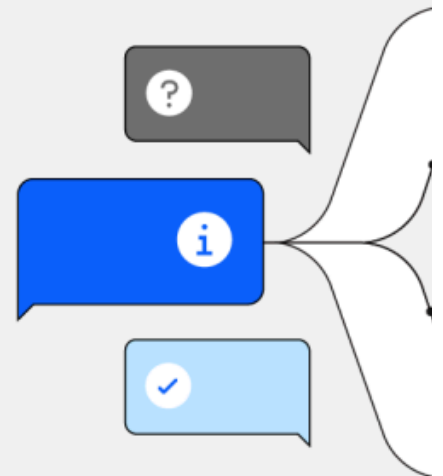


Retrieval Augmented Generation

[Quick start for watsonx SaaS](#)[Deploy watsonx with Cloud Pak for Data](#)

Overview

Let's talk

IBM Product Architecture

Use Cases

Resources

Next steps

Contributors

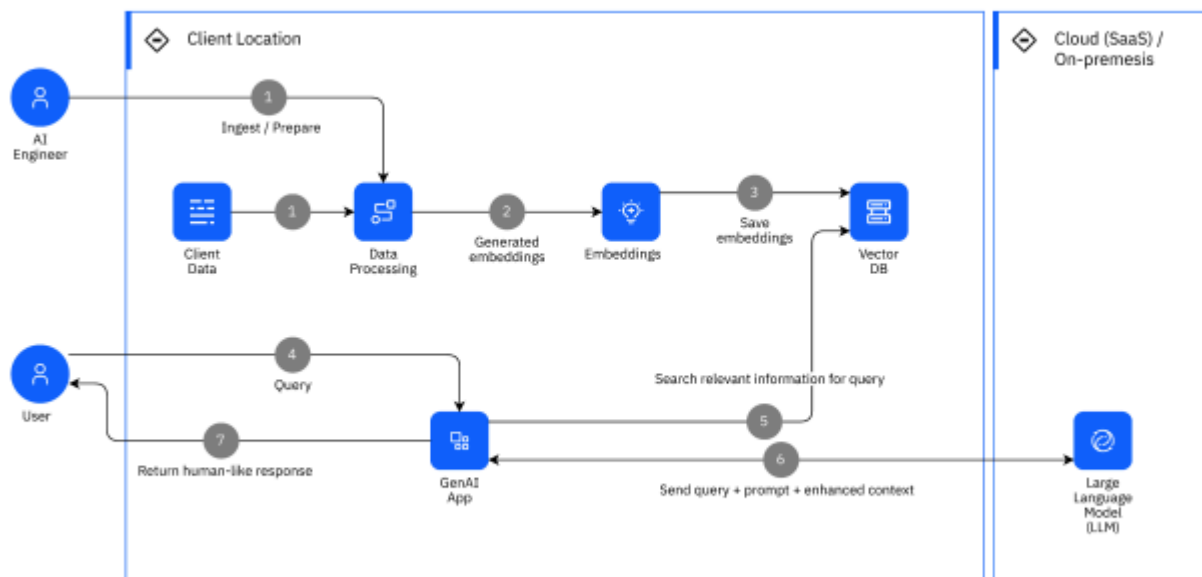
Overview

Large Language Models (LLMs) are often surprisingly knowledgeable about a wide range of topics but they are limited to only the data they were trained on. This means that clients looking to use LLMs with private or proprietary business information cannot use LLMs 'out of the box' to answer questions, generate correspondence, or the like.

Retrieval augmented generation (RAG) is an architectural pattern that enables foundation models to produce factually correct outputs for specialized or proprietary topics that were not part of the model's training data. By augmenting users' questions and prompts with relevant data retrieved from external data sources RAG gives the model 'new' (to the model) facts and details on which to base its response.

Conceptual Architecture

Let's talk



The conceptual architecture of a RAG solution showing the major components and the flow of interactions between them to respond to a user query.

Generative AI architecture patterns

Retrieval augmented generation →

Conversation with agent assist →

Generative search →

Modular reasoning, knowledge, and language →

Document summarization →

Modernization and code generation →

Code generation for Ansible →

Code generation for Z →

AI Augmented Software Development with Agents →

Let's talk

The RAG pattern, shown in the diagram below, is made up of two parts: data embedding during build time, and user prompting (or returning search results) during runtime.

1. An AI Engineer prepares the client data (for example, procedure manuals, product documentation, or help desk tickets, etc.) during Data Preprocessing. Client data is transformed and/or enriched to make it suitable for model augmentation. Transformations might include simple format conversions such as converting PDF documents to text, or more complex transformations such as translating complex table structures into if-then type statements. Enrichment may include expanding common abbreviations, adding meta-data such as currency information, and other additions to improve the relevancy of search results.
2. An embedding model is used to convert the source data into a series of vectors that represent the words in the client data. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. The embeddings are stored as passages (called chunks) of client data, think of sub-sections or paragraphs, to make it easier to find information.
3. The generated embeddings are stored in a vector database. Any data source that supports 'fuzzy' queries that return results based on likely relevance, for example watsonx Discovery, can be used in the RAG architecture but the most common implementation uses a vector database such as Milvus, FAISS, or Chroma.

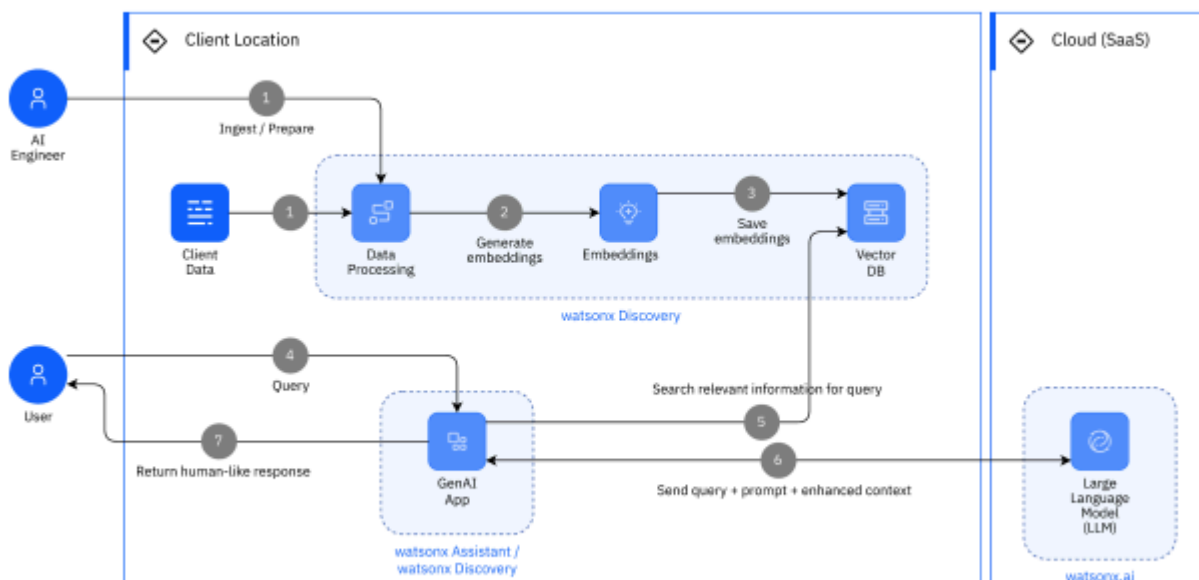
The system is now ready for use by end-users.

4. End-users interact with a GenAI enabled application and enter a query.
5. The GenAI application receives the query, and performs a search on the vector database to obtain the top most (we call this top K) pieces of information that most closely match the user's query. For example, if the user's query is "What is the daily withdrawal limit on the MaxSavers account", the search may return passages such as "The MaxSavers account is...", "Daily withdrawal limits are..." and "...account limits...".
6. The top passages, along with a prompt curated for the specific application sent to the LLM.

Let's talk

7. The LLM returns a human-like response based on the user's query, prompt, and context information which is presented to the end-user.

IBM Product Architecture



An illustration of how IBM watsonx Discovery, IBM watsonx Assistant, and the SaaS version of watsonx.ai realize the RAG solution architecture.

The mapping of the IBM Watson and watsonx family of products to the RAG pattern is shown in the diagram above.

watsonx Discovery implements the pre-processing, embedding generation, and relevancy storage and retrieval functions of the pattern. For certain types of solutions, watsonx Discovery can also be used as the front-end generative AI application for users. Beyond simply replacing a vector database, watsonx Discovery offers out-of-the-box NLP enrichments including entity

Let's talk

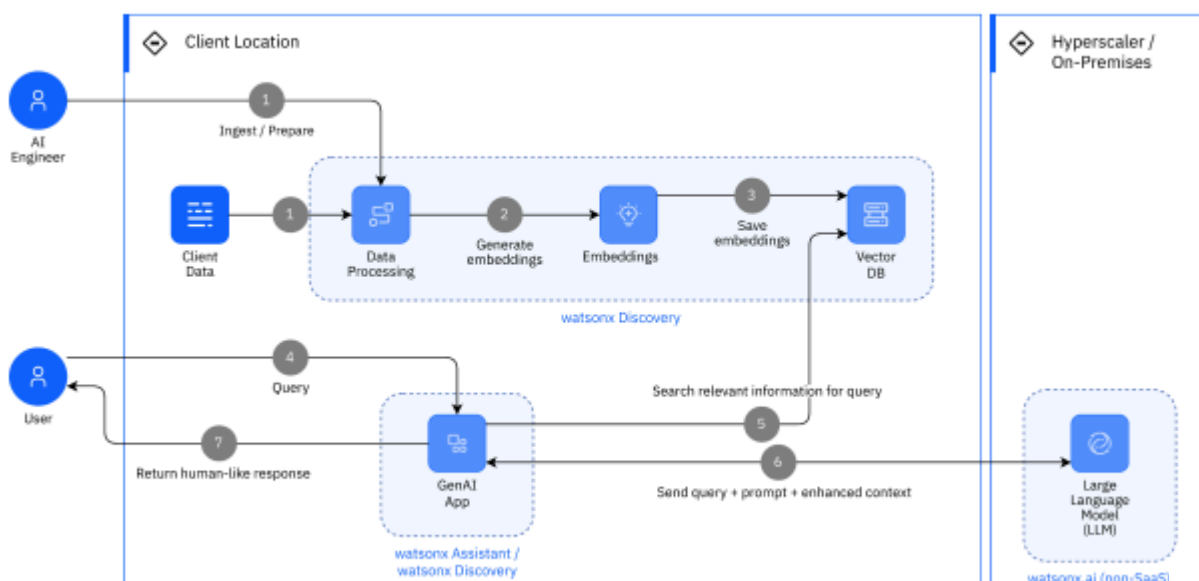
extraction, sentiment analysis, emotion analysis, keyword extractions, category classification, concept tagging and others.

For chat solutions, watsonx Assistant provides the user interface and also conversational capabilities such as remembering the subject of previous queries. For example, if a user asks "Tell me about the Toast-o-matic" and then "How much is it?" watsonx Assistant knows that "it" in the last query refers to the toaster in the first.

Finally, watsonx.ai provides a selection of large language models clients can chose from in a cloud hosting environment. With watsonx.ai, clients can train, validate, tune and deploy generative AI, foundation models and machine learning capabilities with ease and build AI applications in a fraction of the time with a fraction of the data.

On-premise / private deployments

Some clients do not have watsonx.ai available in their local region, or may have security concerns or regulatory requirements that prevent them from using the watsonx.ai SaaS solution. For these clients, we offer watsonx.ai as a set of containerized services that can be deployed on Red Hat Openshift running within the clients' data centers, or within a virtual private cloud (VPC) within a cloud-service provider's infrastructure.

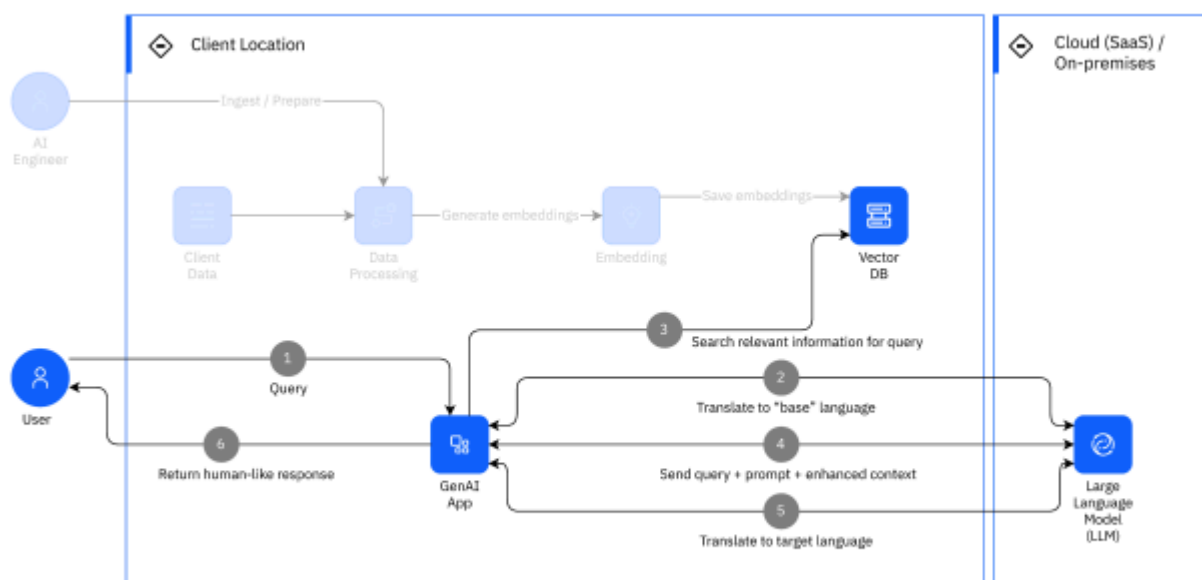


Let's talk

An illustration of how IBM watsonx Discovery, IBM watsonx Assistant, and watsonx.ai can be deployed on-premises to realize the RAG solution architecture.

Multiple language support

The majority of LLMs are trained on English language dominant text that contains a small percentage of text in other, often Western European, languages. For applications requiring multi-lingual or localized language support can implement a pre- and post-query translation step to translate inputs to the pre-processed documents' 'base' language, English for example, and translating the model outputs to the target language, eg. Spanish. This approach is shown in the diagram below.



A walkthrough of the RAG solution architecture illustrating the component interactions and flows to enable support multiple languages.

Let's talk

This approach alters the base RAG pattern as follows (setting the embedding generation steps aside):

1. A user enters a query in a language that differs from the dominant language of the pre-processed documentation. For example, a query in Spanish and an English dominant documentation base.
2. The generative AI application prompts a large language model to translate the user query to the language of the documentation base. In our example, from Spanish to English.
3. The translated query is used to retrieve the top K passages of information that are most relevant to the user's query.
4. The translated query and the retrieved context are sent to the LLM to generate a response.
5. The generative AI application again uses a large language model to translate the generated response to the user's target language. In our example, from English to Spanish.
6. The translated response, in Spanish, is presented to the end-user.

Experience suggests that 80% or higher accuracy, depending on the context and types of queries being submitted, in non-base-language results can be achieved using this approach. Emerging multi-language models, trained on larger percentages of different languages, are expected to achieve even higher accuracy.

Use Cases

RAG is a candidate solution for any business scenario where there is a large body of documentation and business rules that a user must consult to provide authoritative answers. It is also a strong solution for infusing LLM-based chatbots with proprietary or domain-specific knowledge and preventing hallucinations.

Candidate uses include:

- **Insurance underwriting and claims adjudication.** RAG has many potential applications within the insurance industry. Underwriters and brokers require knowledge of thousands of pages of documentation covering the terms and conditions of policies. Let's talk

conditions of hundreds of insurance products. Similarly, claims adjudicators may be required to have deep knowledge of the same documentation, as well as contracts with overrides and additional terms specific to individual clients. The RAG architecture pattern can serve as the architecture 'backbone' of solutions to assist underwrites, brokers, and adjusters with querying product and contract documentation to better respond to client queries and improve process productivity.

- **Call center agent support.** Call center agents require deep knowledge of potentially hundreds of products and services, as well as commonly occurring product issues and their resolution. The RAG pattern is a strong architecture foundation on which to create solutions to assist agents in quickly finding answers to client requests.
- **Customer chatbots.** RAG is strong enabler for creating customer-facing chatbots to answer questions. Combining the natural language abilities of Large Language Models and the enterprise-specific responses of RAG can deliver a compelling, conversational customer experience. Note that RAG on its own only delivers question-and-answer capabilities; it does not have the ability to 'transact', ie. interact with enterprise systems to pull information or update records. Additional components must be added to detect user intent and to interact with enterprise systems.
- **Support / helpdesk.** Like call center agents, IT operations and support personnel require deep knowledge of the configuration of complex systems deployments along with knowledge of common and previously seen issues and their resolution. The RAG pattern is a strong architecture foundation on which to create solutions to assist support personnel with quickly finding relevant answers to reported problems and observed issues.

Architecture Decisions and Considerations

Choice of Generation Model	▼
Choice of Embedding Model	▼
Database	▼
Augmentation Method	▼
Data Preprocessing	▼
Evaluation Metrics	▼
File Conversion	

Let's talk

Response Caching	▼
Proprietary, Confidential, and Personal Information	▼
Encrypting Data in Transit	▼
Inter-zone Connectivity	▼

Resources

Quick start: Prompt a foundation model with the retrieval-augmented generation pattern to generate factually accurate output grounded in information in a knowledge base by applying the

IBM Generative AI Architecture


IBM's Generative AI Architecture is the complete IBM Generative AI Architecture in IBM IT Architect Assistant (IIAA), an architecture

Next steps


Let's talk

Talk to our experts about implementing a hybrid cloud deployment pattern.

Contact us



Contact us



More ways to explore

- [Hybrid Cloud Architecture Center](#)

→
- [Diagram tools and templates](#)

→
- [IBM Well-Architected Framework](#)

→

[David Massey](#), [Manav Gupta](#), [Mihai Criveti](#), [Chris Kirby](#), [Pete Nuwayser](#)
Updated: November 28, 2023

Let’s talk