Retrieval Augmented Generation (RAG) in Azure AI Search

08/18/2025

Retrieval Augmented Generation (RAG) is a design pattern that augments the capabilities of a chat completion model like ChatGPT by adding an information retrieval step, incorporating your proprietary enterprise content for answer formulation. For an enterprise solution, it's possible to fully constrain generative AI to your enterprise content.

The decision about which information retrieval system to use is critical because it determines the inputs to the LLM. The information retrieval system should provide:

- Indexing strategies that load and refresh at scale, for all of your content, at the frequency you require.
- Query capabilities and relevance tuning. The system should return *relevant* results, in the short-form formats necessary for meeting the token length requirements of large language model (LLM) inputs.
- Security, global reach, and reliability for both data and operations.
- Integration with embedding models for indexing, and chat models or language understanding models for retrieval.

Azure AI Search is a proven solution for information retrieval in a RAG architecture. It provides indexing and query capabilities, with the infrastructure and security of the Azure cloud. Through code and other components, you can design a comprehensive RAG solution that includes all of the elements for generative AI over your proprietary content.

> **Note**
>
> New to copilot and RAG concepts? Watch **Vector search and state of the art retrieval for Generative AI apps** .

Approaches for RAG with Azure AI Search

Microsoft has several built-in implementations for using Azure AI Search in a RAG solution.

- Azure AI Search, build an agentic retrieval pipeline in a custom solution. The agentic pipeline is designed specifically for the RAG pattern. You write code that calls Azure AI Search APIs designed for chat completion model integration with your indexed contet.
- Azure AI Foundry, use a vector index and retrieval augmentation.
- Azure OpenAI, use a search index with or without vectors.
- Azure Machine Learning, use a search index as a vector store in a prompt flow.

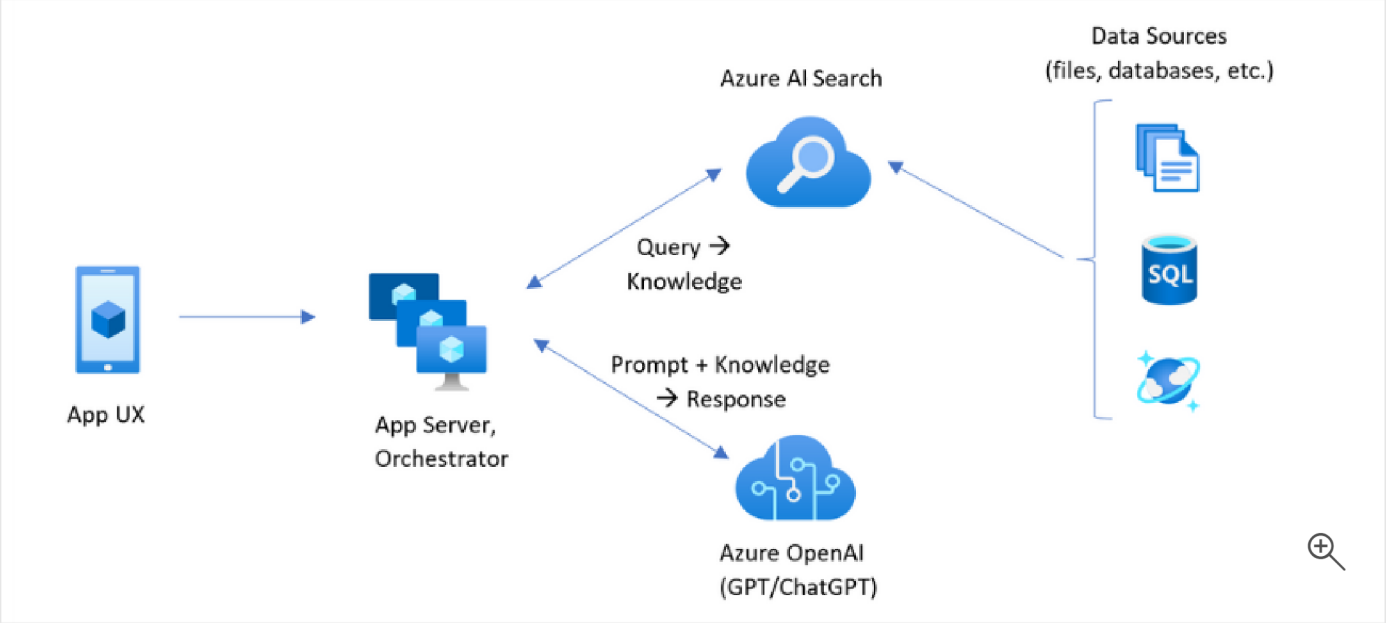
Custom RAG pattern for Azure AI Search

A high-level summary of a RAG pattern based on Azure AI Search looks like this:

- Start with a user question or request (prompt).
- Send it to Azure AI Search to find relevant information.
- Return the top ranked search results to an LLM.
- Use the natural language understanding and reasoning capabilities of the LLM to generate a response to the initial prompt.

Azure AI Search provides inputs to the LLM prompt, but doesn't train the model. In a traditional RAG pattern, there's no extra training. The LLM is pretrained using public data, but it generates responses that are augmented by information from the retriever, in this case, Azure AI Search.

RAG patterns that include Azure AI Search have the elements indicated in the following illustration.



- App UX (web app) for the user experience
- App server or orchestrator (integration and coordination layer)

- Azure AI Search (information retrieval system)
- Azure OpenAI (LLM for generative AI)

The web app provides the user experience, providing the presentation, context, and user interaction. Questions or prompts from a user start here. Inputs pass through the integration layer, going first to information retrieval to get the search results, but also go to the LLM to set the context and intent.

The app server or orchestrator is the integration code that coordinates the handoffs between information retrieval and the LLM. Common solutions include Azure Semantic Kernel or LangChain to coordinate the workflow. LangChain integrates with Azure AI Search, making it easier to include Azure AI Search as a retriever in your workflow. LlamaIndex and Semantic Kernel are other options.

The information retrieval system provides the searchable index, query logic, and the payload (query response). The search index can contain vectors or nonvector content. Although most samples and demos include vector fields, it's not a requirement. The query is executed using the existing search engine in Azure AI Search, which can handle keyword (or term) and vector queries. The index is created in advance, based on a schema you define, and loaded with your content that's sourced from files, databases, or storage.

The LLM receives the original prompt, plus the results from Azure AI Search. The LLM analyzes the results and formulates a response. If the LLM is ChatGPT, the user interaction might consist of multiple conversation turns. An Azure solution most likely uses Azure OpenAI, but there's no hard dependency on this specific service.

Except for features currently in previewAzure AI Search doesn't provide native LLM integration for prompt flows or chat preservation, so you need to write code that handles orchestration and state. You can review demo source (Azure-Samples/azure-search-openai-demo), updated for agentic retrieval, for a blueprint of what a full solution entails. We also recommend Azure AI Foundry to create RAG-based Azure AI Search solutions that integrate with LLMs.

Searchable content in Azure AI Search

In Azure AI Search, all searchable content is stored in a search index that's hosted on your search service. A search index is designed for fast queries with millisecond response times, so its internal data structures exist to support that objective. To that end, a search index stores *indexed content*, and not whole content files like entire PDFs or images. Internally, the data structures include inverted indexes of tokenized text, vector indexes for embeddings, and unaltered plain text for

cases where verbatim matching is required (for example, in filters, fuzzy search, regular expression queries).

When you set up the data for your RAG solution, you use the features that create and load an index in Azure AI Search. An index includes fields that duplicate or represent your source content. An index field might be simple transference (a title or description in a source document becomes a title or description in a search index), or a field might contain the output of an external process, such as vectorization or skill processing that generates a representation or text description of an image.

Since you probably know what kind of content you want to search over, consider the indexing features that are applicable to each content type:

Expand table

Content type	Indexed as	Features
text	tokens, unaltered text	Indexers can pull plain text from other Azure resources like Azure Storage and Cosmos DB. You can also push any JSON content to an index. To modify text in flight, use analyzers and normalizers to add lexical processing during indexing. Synonym maps are useful if source documents are missing terminology that might be used in a query.
text	vectors [1]	Text can be chunked and vectorized in an indexer pipeline, or handled externally and then indexed as vector fields in your index.
image	tokens, unaltered text [2]	Skills for OCR and Image Analysis can process images for text recognition or image characteristics. Skills have an indexer requirement.
image	vectors [1]	Images can be vectorized in an indexer pipeline, or handled externally for a mathematical representation of image content and then indexed as vector fields in your index. You can use Azure AI Vision multimodal or an open source model like OpenAI CLIP to vectorize text and images in the same embedding space.

[1] Azure AI Search provides integrated data chunking and vectorization, but you must take a dependency on indexers and skillsets. If you can't use an indexer, Microsoft's Semantic Kernel or other community offerings can help you with a full stack solution. For code samples showing both approaches, see azure-search-vectors repo .

[2] Image descriptions are converted to searchable text and added to the index. The images themselves are not stored in the index. Skills are built-in support for applied AI. For OCR and

Image Analysis, the indexing pipeline makes an internal call to the Azure AI Vision APIs. These skills pass an extracted image to Azure AI for processing, and receive the output as text that's indexed by Azure AI Search. Skills are also used for integrated data chunking (Text Split skill) and integrated embedding (skills that call Azure AI Vision multimodal, Azure OpenAI, and models in the Azure AI Foundry model catalog.)

Vectors provide the best accommodation for dissimilar content (multiple file formats and languages) because content is expressed universally in mathematic representations. Vectors also support similarity search: matching on the coordinates that are most similar to the vector query. Compared to keyword search (or term search) that matches on tokenized terms, similarity search is more nuanced. It's a better choice if there's ambiguity or interpretation requirements in the content or in queries.

Content retrieval in Azure AI Search

Once your data is in a search index, you use the query capabilities of Azure AI Search to retrieve content.

In a non-RAG pattern, queries make a round trip from a search client. The query is submitted, it executes on a search engine, and the response returned to the client application. The response, or search results, consist exclusively of the verbatim content found in your index.

In a RAG pattern, queries and responses are coordinated between the search engine and the LLM. A user's question or query is forwarded to both the search engine and to the LLM as a prompt. The search results come back from the search engine and are redirected to an LLM. The response that makes it back to the user is generative AI, either a summation or answer from the LLM.

There's no query type in Azure AI Search - not even semantic or vector search - that composes new answers. Only the LLM provides generative AI. Here are the capabilities in Azure AI Search that are used to formulate queries:

Expand table

Query feature	Purpose	Why use it
Simple or full Lucene syntax	Query execution over text and nonvector numeric content	Full text search is best for exact matches, rather than similar matches. Full text search queries are ranked using the BM25 algorithm and support relevance tuning

Query feature	Purpose	Why use it
		through scoring profiles. It also supports filters and facets.
Filters and facets	Applies to text or numeric (nonvector) fields only. Reduces the search surface area based on inclusion or exclusion criteria.	Adds precision to your queries.
Semantic ranker	Re-ranks a BM25 result set using semantic models. Produces short-form captions and answers that are useful as LLM inputs.	Easier than scoring profiles, and depending on your content, a more reliable technique for relevance tuning.
Vector search	Query execution over vector fields for similarity search, where the query string is one or more vectors.	Vectors can represent all types of content, in any language.
Hybrid search	Combines any or all of the above query techniques. Vector and nonvector queries execute in parallel and are returned in a unified result set.	The most significant gains in precision and recall are through hybrid queries.
Agentic search (preview)	Parallel query execution pipeline of multiple subqueries, returning a response designed for RAG workloads and agent consumer. Queries can be vector or keyword search. Semantic ranking ensures the best results of subquery are included in the final response structure. **This is the recommended approach for RAG patterns based on Azure AI Search**.	

Structure the query response

A query's response provides the input to the LLM, so the quality of your search results is critical to success. Results are a tabular row set. The composition or structure of the results depends on:

- Fields that determine which parts of the index are included in the response.
- Rows that represent a match from index.

Fields appear in search results when the attribute is "retrievable". A field definition in the index schema has attributes, and those determine whether a field is used in a response. Only "retrievable" fields are returned in full text or vector query results. By default all "retrievable" fields are returned, but you can use "select" to specify a subset. Besides "retrievable", there are no restrictions on the field. Fields can be of any length or type. Regarding length, there's no maximum field length limit in Azure AI Search, but there are limits on the size of an API request.

Rows are matches to the query, ranked by relevance, similarity, or both. By default, results are capped at the top 50 matches for full text search or k-nearest-neighbor matches for vector search. You can change the defaults to increase or decrease the limit up to the maximum of 1,000 documents. You can also use top and skip paging parameters to retrieve results as a series of paged results.

Maximize relevance and recall

When you're working with complex processes, a large amount of data, and expectations for millisecond responses, it's critical that each step adds value and improves the quality of the end result. On the information retrieval side, *relevance tuning* is an activity that improves the quality of the results sent to the LLM. Only the most relevant or the most similar matching documents should be included in results.

Here are some tips for maximizing relevance and recall:

- Hybrid queries that combine keyword (nonvector) search and vector search give you maximum recall when the inputs are the same. In a hybrid query, if you double down on the same input, a text string and its vector equivalent generate parallel queries for keywords and similarity search, returning the most relevant matches from each query type in a unified result set.

- Hybrid queries can also be expansive. You can run similarity search over verbose chunked content, and keyword search over names, all in the same request.

- Relevance tuning is supported through:

  - Scoring profiles that boost the search score if matches are found in a specific search field or on other criteria.

  - Semantic ranker that re-ranks an initial results set, using semantic models from Bing to reorder results for a better semantic fit to the original query.

  - Query parameters for fine-tuning. You can boost the importance of vector queries or adjust the amount of BM25-ranked results in a hybrid query response. You can also set minimum thresholds to exclude low scoring results from a vector query.

In comparison and benchmark testing, hybrid queries with text and vector fields, supplemented with semantic ranking, produce the most relevant results.

Example code for a RAG workflow

The following Python code demonstrates the essential components of a basic RAG workflow in Azure AI Search. You need to set up the clients, define a system prompt, and provide a query. The prompt tells the LLM to use just the results from the query, and how to return the results. For more steps based on this example, see this RAG quickstart.

> **Note**
>
> For the Azure Government cloud, modify the API endpoint on the token provider to `"https://cognitiveservices.azure.us/.default"`.

Python

```
# Set up the query for generating responses
from azure.identity import DefaultAzureCredential
from azure.identity import get_bearer_token_provider
from azure.search.documents import SearchClient
from openai import AzureOpenAI

credential = DefaultAzureCredential()
token_provider = get_bearer_token_provider(credential,
"https://cognitiveservices.azure.com/.default")
openai_client = AzureOpenAI(
    api_version="2024-06-01",
    azure_endpoint=AZURE_OPENAI_ACCOUNT,
    azure_ad_token_provider=token_provider
)

search_client = SearchClient(
    endpoint=AZURE_SEARCH_SERVICE,
    index_name="hotels-sample-index",
    credential=credential
)

# This prompt provides instructions to the model.
# The prompt includes the query and the source, which are specified further down in
the code.
GROUNDED_PROMPT="""
You are a friendly assistant that recommends hotels based on activities and ameni-
ties.
Answer the query using only the sources provided below in a friendly and concise bul-
leted manner.
Answer ONLY with the facts listed in the list of sources below.
If there isn't enough information below, say you don't know.
Do not generate answers that don't use the sources below.
```

```
Query: {query}
Sources:\n{sources}
"""

# The query is sent to the search engine, but it's also passed in the prompt
query="Can you recommend a few hotels near the ocean with beach access and good
views"

# Retrieve the selected fields from the search index related to the question
search_results = search_client.search(
    search_text=query,
    top=5,
    select="Description,HotelName,Tags"
)
sources_formatted = "\n".join([f'{document["HotelName"]}:{document["Description"]}:
{document["Tags"]}' for document in search_results])

response = openai_client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": GROUNDED_PROMPT.format(query=query, sources=sources_formatted)
        }
    ],
    model="gpt-35"
)

print(response.choices[0].message.content)
```

Integration code and LLMs

A RAG solution that includes Azure AI Search can leverage built-in data chunking and vectorization capabilities, or you can build your own using platforms like Semantic Kernel, LangChain, or LlamaIndex.

Notebooks in the demo repository are a great starting point because they show patterns for LLM integration. Much of the code in a RAG solution consists of calls to the LLM so you need to develop an understanding of how those APIs work, which is outside the scope of this article.

How to get started

There are many ways to get started, including code-first solutions and demos.

Docs

- Try this RAG quickstart for a demonstration of query integration with chat models over a search index.
- Try this agentic retrieval quickstart to walk through the new and recommended approach for RAG.
- Tutorial: How to build a RAG solution in Azure AI Search for focused coverage on the features and pattern for RAG solutions that obtain grounding data from a search index.
- Review indexing concepts and strategies to determine how you want to ingest and refresh data. Decide whether to use vector search, keyword search, or hybrid search. The kind of content you need to search over, and the type of queries you want to run, determines index design.
- Review creating queries to learn more about search request syntax and requirements.

> **Note**
>
> Some Azure AI Search features are intended for human interaction and aren't useful in a RAG pattern. Specifically, you can skip features like autocomplete and suggestions. Other features like facets and orderby might be useful, but would be uncommon in a RAG scenario.

See also

- RAG Experiment Accelerator
- Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models
- Azure Cognitive Search and LangChain: A Seamless Integration for Enhanced Vector Search Capabilities