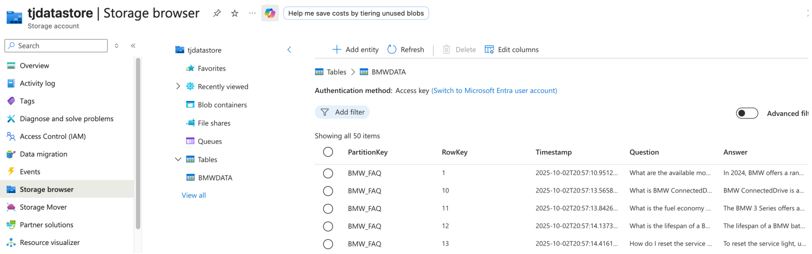
RAG + AZURE AI

1. Set up Storage Account

2. Store Table in Azure Table Storage



tjdatastore | Storage browser
Storage account

Help me save costs by tiering unused blobs

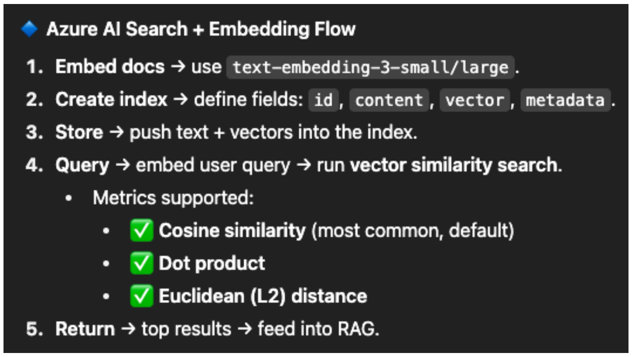
Tables > BMWDATA

Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 50 items

○	PartitionKey	RowKey	Timestamp	Question	Answer
○	BMW_FAQ	1	2025-10-02T20:57:10.9512...	What are the available mo...	In 2024, BMW offers a ran...
○	BMW_FAQ	10	2025-10-02T20:57:13.5658...	What is BMW ConnectedD...	BMW ConnectedDrive is a...
○	BMW_FAQ	11	2025-10-02T20:57:13.8426...	What is the fuel economy ...	The BMW 3 Series offers a...
○	BMW_FAQ	12	2025-10-02T20:57:14.1373...	What is the lifespan of a B...	The lifespan of a BMW bat...
○	BMW_FAQ	13	2025-10-02T20:57:14.4161...	How do I reset the service ...	To reset the service light, u...

3. Azure AI Search → Load Document



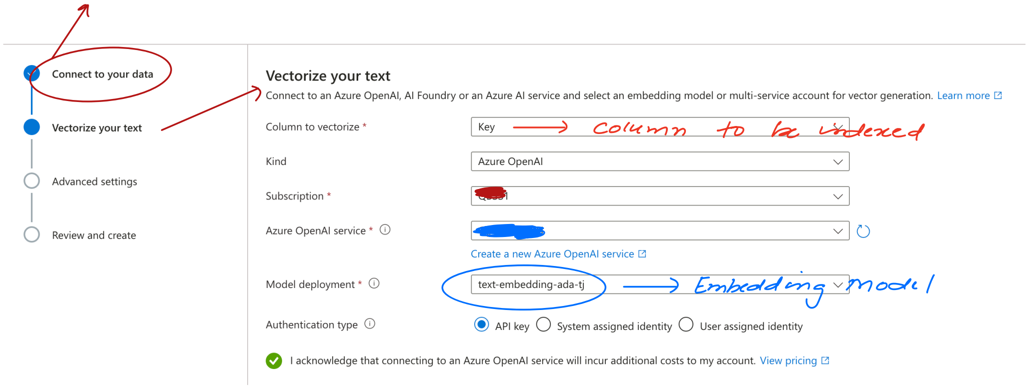
🔷 **Azure AI Search + Embedding Flow**

1. **Embed docs** → use `text-embedding-3-small/large`.
2. **Create index** → define fields: `id`, `content`, `vector`, `metadata`.
3. **Store** → push text + vectors into the index.
4. **Query** → embed user query → run **vector similarity search**.
   - Metrics supported:
     - ✅ **Cosine similarity** (most common, default)
     - ✅ **Dot product**
     - ✅ **Euclidean (L2) distance**
5. **Return** → top results → feed into RAG.

Go to Azure AI Search - Import Data

Choose RAG / Keyword

(BMW Data)



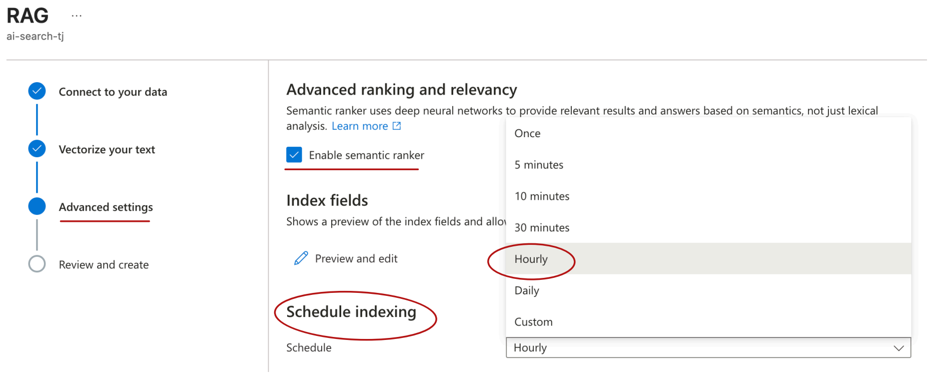
Vectorize your text

Connect to an Azure OpenAI, AI Foundry or an Azure AI service and select an embedding model or multi-service account for vector generation. Learn more

Column to vectorize *	Key → Column to be indexed
Kind	Azure OpenAI
Subscription *	[redacted]
Azure OpenAI service *	[redacted]
	Create a new Azure OpenAI service
Model deployment *	text-embedding-ada-tj → Embedding model
Authentication type	● API key ○ System assigned identity ○ User assigned identity

✅ I acknowledge that connecting to an Azure OpenAI service will incur additional costs to my account. View pricing

Imp: So here I will be assigning Text Embedding Model & Index Type



RAG ···
ai-search-tj

Advanced ranking and relevancy

Semantic ranker uses deep neural networks to provide relevant results and answers based on semantics, not just lexical analysis. Learn more

☑ Enable semantic ranker

Index fields

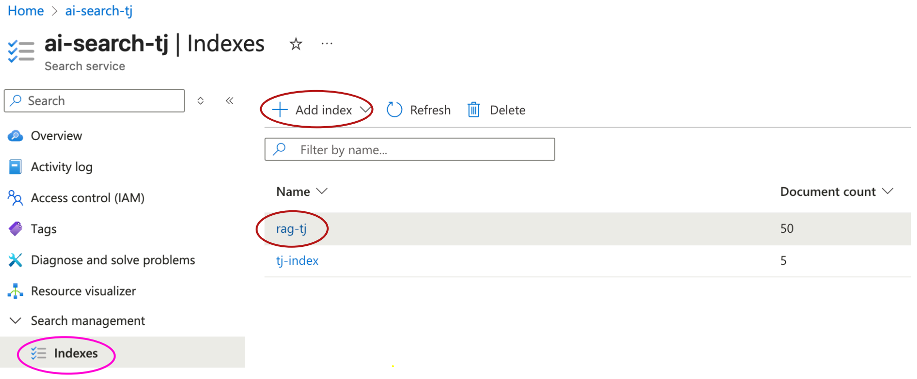
Shows a preview of the index fields and allov

Schedule indexing

Schedule: Hourly

4. Set env Variables for AI Search

- AI Search Acc. ai-seach-tj
- Index Name rag-tj
- AI Search Api Key. * * *

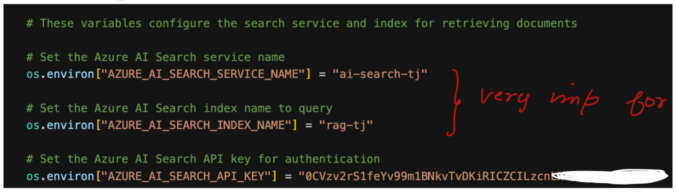


Home > ai-search-tj

ai-search-tj | Indexes ☆ ···
Search service

Name	Document count
rag-tj	50
tj-index	5

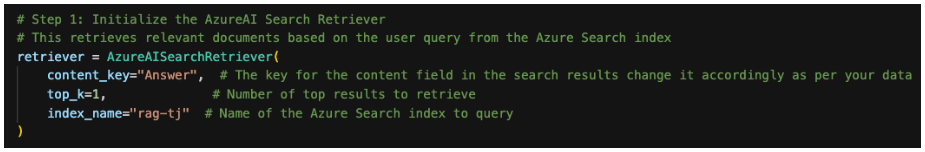
Code Example



```
# These variables configure the search service and index for retrieving documents

# Set the Azure AI Search service name
os.environ["AZURE_AI_SEARCH_SERVICE_NAME"] = "ai-search-tj"

# Set the Azure AI Search index name to query
os.environ["AZURE_AI_SEARCH_INDEX_NAME"] = "rag-tj"

# Set the Azure AI Search API key for authentication
os.environ["AZURE_AI_SEARCH_API_KEY"] = "0CVzv2rS1feYv99m1BNkvTvDKiRICZCILzcn[redacted]
```

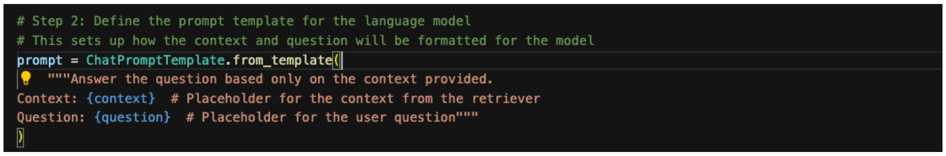
} very imp for Retriever

1. Azure AI Retriever



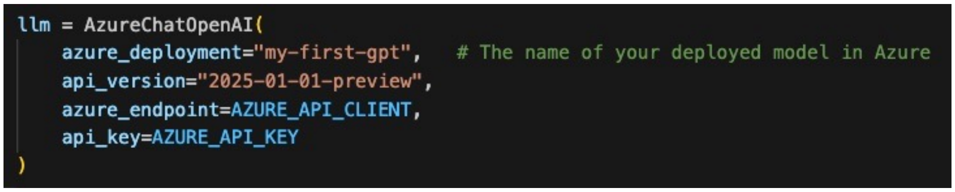
```
# Step 1: Initialize the AzureAI Search Retriever
# This retrieves relevant documents based on the user query from the Azure Search index
retriever = AzureAISearchRetriever(
    content_key="Answer",  # The key for the content field in the search results change it accordingly as per your data
    top_k=1,              # Number of top results to retrieve
    index_name="rag-tj"  # Name of the Azure Search index to query
)
```

2. Prompt



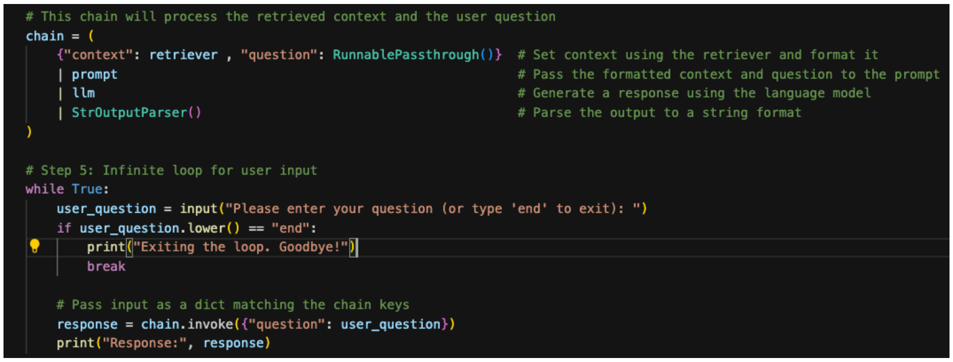
```python
# Step 2: Define the prompt template for the language model
# This sets up how the context and question will be formatted for the model
prompt = ChatPromptTemplate.from_template(
    """Answer the question based only on the context provided.
Context: {context}  # Placeholder for the context from the retriever
Question: {question}  # Placeholder for the user question"""
)
```

3. LLm



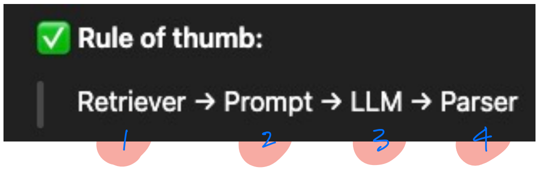
```python
llm = AzureChatOpenAI(
    azure_deployment="my-first-gpt",   # The name of your deployed model in Azure
    api_version="2025-01-01-preview",
    azure_endpoint=AZURE_API_CLIENT,
    api_key=AZURE_API_KEY
)
```

4. Chain → Context, Prompt, LLm



```python
# This chain will process the retrieved context and the user question
chain = (
    {"context": retriever , "question": RunnablePassthrough()}  # Set context using the retriever and format it
    | prompt                                                     # Pass the formatted context and question to the prompt
    | llm                                                        # Generate a response using the language model
    | StrOutputParser()                                          # Parse the output to a string format
)

# Step 5: Infinite loop for user input
while True:
    user_question = input("Please enter your question (or type 'end' to exit): ")
    if user_question.lower() == "end":
        print("Exiting the loop. Goodbye!")
        break

    # Pass input as a dict matching the chain keys
    response = chain.invoke({"question": user_question})
    print("Response:", response)
```

Imp. The order is imp in Chain



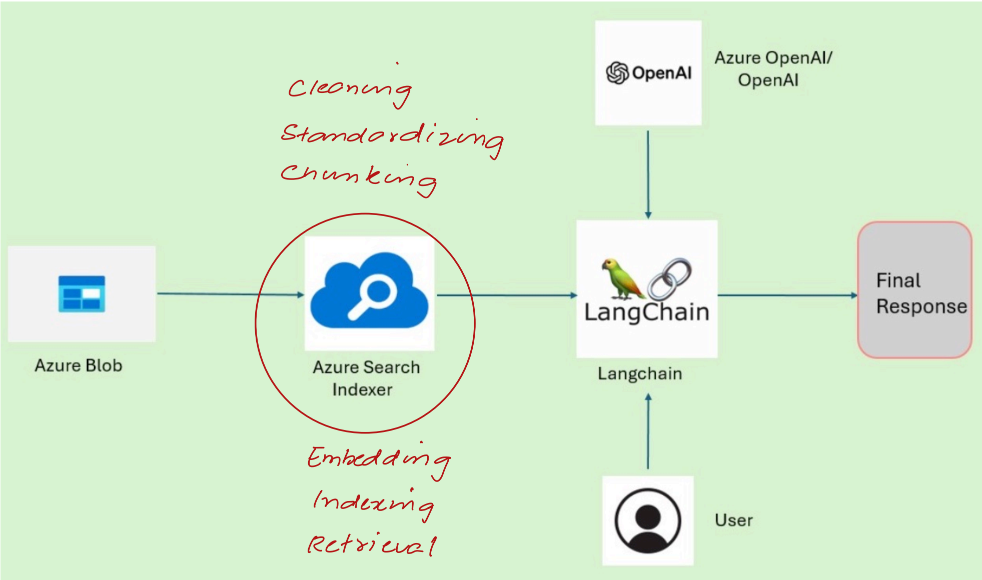
✅ **Rule of thumb:**

> Retriever → Prompt → LLM → Parser

1 2 3 4

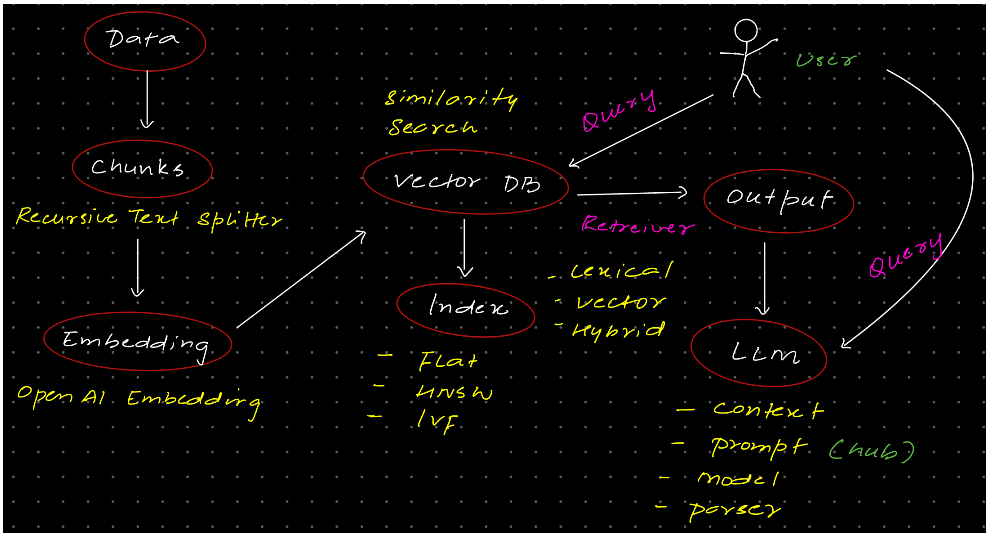
AZURE AI SEARCH + LANGCHAIN

AI Search makes process really easy.



RAG + LANGHAIN

Complex process



Imp. Azure AI Search makes process easy.