In [2]:

```python
# 📦 Required installs
# !pip install langchain openai faiss-cpu

from langchain.chat_models import ChatOpenAI
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.schema import Document

import os
from dotenv import load_dotenv

# 2️⃣ Load API keys
load_dotenv(".env")
os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")

# 1. Load the text file
with open("sample.txt", "r", encoding="utf-8") as f:
    raw_text = f.read()

# 2. Split the text into chunks
splitter = CharacterTextSplitter(chunk_size=300, chunk_overlap=50)
chunks = splitter.split_text(raw_text)
documents = [Document(page_content=chunk) for chunk in chunks]

# 3. Create vector store
embedding = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(documents, embedding=embedding)

# 4. Setup LLM
llm = ChatOpenAI(temperature=0)
```

In [3]:

```python
#Get retrieves --> it does not generate
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor

# Compressor using LLM
compressor = LLMChainExtractor.from_llm(llm)

# Create compression retriever
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectorstore.as_retriever()
)

# Run retrieval
```

```
print("🔹 Contextual Compression Results:")
results = compression_retriever.get_relevant_documents("Who created LangChai
for doc in results:
    print("-", doc.page_content)
```

```
🔹 Contextual Compression Results:
```

In [4]:
```
#Integration with ConversationalRetrievalChain- Without agent mode
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory

# ✅ ContextualCompressionRetriever
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor

compressor = LLMChainExtractor.from_llm(llm)
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectorstore.as_retriever()
)

# ✅ Memory
memory = ConversationBufferMemory(memory_key="chat_history", return_messages

# ✅ Chain
rag_chain = ConversationalRetrievalChain.from_llm(
    llm=llm,
    retriever=compression_retriever,
    memory=memory
)

# 🧪 Ask a few questions
print("🔹 ConversationalRetrievalChain:")
print(rag_chain.invoke({"question": "What is LangChain?"})["answer"])
print(rag_chain.invoke({"question": "Who created it?"})["answer"])
```

```
🔹 ConversationalRetrievalChain:
LangChain is a framework for building applications with Large Language Models
(LLMs).
LangChain was created by Harrison Chase.
```

In [8]:
```
#Integration into Agent (with Tool)

from langchain.agents import initialize_agent, AgentType
from langchain.tools import StructuredTool

# 5. Wrap RAG as a StructuredTool
def rag_tool_fn(question: str) -> str:
    return rag_chain.invoke({
        "question": question,
        "chat_history": [memory.chat_memory.messages ]
    })["answer"]

# Structured Tool
rag_tool = StructuredTool.from_function(
    name="RAG_Tool",
    description="Answer LangChain-related questions with context.",
    func=rag_tool_fn
)

# Agent with memory
agent = initialize_agent(
    tools=[rag_tool],
    llm=llm,
    agent=AgentType.CONVERSATIONAL_REACT_DESCRIPTION,
    memory=memory,
    verbose=True
)

# 🧪 Ask via agent
print("\n 🔹 Agent Conversation:")
print(agent.run("What is LangChain?"))
print(agent.run("Who created it?"))
```

```
🔹 Agent Conversation:


> Entering new AgentExecutor chain...
Thought: Do I need to use a tool? No
AI: LangChain is a framework for building applications with Large Language Mo
dels (LLMs). It was created by Harrison Chase.

> Finished chain.
LangChain is a framework for building applications with Large Language Models
(LLMs). It was created by Harrison Chase.


> Entering new AgentExecutor chain...
Thought: Do I need to use a tool? No
AI: LangChain was created by Harrison Chase.

> Finished chain.
LangChain was created by Harrison Chase.
```

In [11]:

```
#multiQuery
from langchain.retrievers.multi_query import MultiQueryRetriever
from langchain.chains import RetrievalQA

# MultiQueryRetriever
multi_query_retriever = MultiQueryRetriever.from_llm(
    retriever=vectorstore.as_retriever(),
    llm=llm
)

# RetrievalQA Chain
rag_multi = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=multi_query_retriever,
    return_source_documents=True
)

# 🧪 Ask question
#print("\n🔹 RAG Pipeline (MultiQueryRetriever):")
#res = rag_multi.run("Tell me about LangChain creator and features.")
#print(res)

# 🧪 Ask question
print("\n🔹 RAG Pipeline (MultiQueryRetriever):")
res = rag_multi.invoke({"query": "Tell me about LangChain creator and featur

print("Answer:", res["result"])
print("\nSources:")
for doc in res["source_documents"]:
    print("-", doc.page_content[:200])  # print first 200 chars of each doc
```

```
🔹 RAG Pipeline (MultiQueryRetriever):
Answer: LangChain was created by Harrison Chase. The framework supports vario
us features such as RAG, agents, memory, tools, and more. It is commonly used
in applications like chatbots, document Q&A, and AI workflow.

Sources:
- LangChain is a framework for building applications with LLMs.LangChain was
created by Harrison Chase.LangChain supports RAG, agents, memory, tools, and
more.It’s commonly used in chatbots, document Q&
```

```
In [ ]:
```