

Retriever Type	Purpose	Ideal Usage Scenario
VectorStoreRetriever	Retrieve documents based on vector similarity search (e.g. FAISS, Chroma, Pinecone)	Basic RAG (Retrieval-Augmented Generation) setup for fetching semantically relevant documents
ContextualCompressionRetriever	Compresses retrieved docs using an LLM to return only the most relevant parts	When using large context docs (PDFs, transcripts) where only a portion of each doc is relevant
MultiQueryRetriever	Generates multiple rephrased queries to improve retrieval coverage	When a user query can be ambiguous or phrased differently; improves diversity of retrieved documents
ParentDocumentRetriever	Retrieves a full parent document instead of the split chunk	When a chunk is matched but full document context is required (e.g., blog, article, contract analysis)
BM25Retriever	Keyword-based retriever using classic BM25 algorithm	When you want to retrieve based on exact keyword matches (useful for legal, medical, or short documents with precise language)
EnsembleRetriever	Combines multiple retrievers (e.g., vector + BM25) with weighted scoring	When you want to merge semantic and keyword-based search for better hybrid performance
TimeWeightedVectorStoreRetriever	VectorStoreRetriever + time-based decay to prefer recent documents	Best for chatbot memory or news search where recency of information is important
TavilySearchAPIRetriever	Uses external web search engine via Tavily API to retrieve live web data	When querying real-world events, breaking news, or updated information outside your local docs or vector database

```
In [1]: #1)VectorStoreRetriever Vector-based      Embedding similarity search      Generat
# ConversationalRetrievalChain

from langchain.chat_models import ChatOpenAI
from langchain.embeddings import OpenAIEMBEDDINGS
from langchain.vectorstores import FAISS
from langchain.chains import ConversationalRetrievalChain
from langchain.agents import initialize_agent, AgentType
from langchain.tools import StructuredTool
from langchain.memory import ConversationBufferMemory
import os

# 1. Setup
```

```
import os
from dotenv import load_dotenv

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

llm = ChatOpenAI(temperature=0)

# 2. Vector DB
with open("sample.txt", "r", encoding="utf-8") as f:
    text_data = f.read()

# 🧠 Split the text into smaller chunks
from langchain.text_splitter import CharacterTextSplitter
splitter = CharacterTextSplitter(separator="\n", chunk_size=300, chunk_overlap=0)
texts = splitter.split_text(text_data)

embedding = OpenAIEmbeddings()
vectorstore = FAISS.from_texts(texts, embedding)
retriever = vectorstore.as_retriever()

# 3. Conversational RAG chain
rag_chain = ConversationalRetrievalChain.from_llm(
    llm=llm,
    retriever=retriever,
    return_source_documents=False
)

# 4. Memory for chat history
memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)

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

rag_tool = StructuredTool.from_function(
    name="RAG_QA",
    description="Use this to answer questions about LangChain.",
    func=rag_tool_fn
)

# 6. Create agent
agent = initialize_agent(
    tools=[rag_tool],
    llm=llm,
    agent=AgentType.CONVERSATIONAL_REACT_DESCRIPTION,
    verbose=True,
    memory=memory,
    handle_parsing_errors=True
)
```

```
# 7. Run conversation
print("1 First question")
res1 = agent.run("What is LangChain?")
print("Answer:", res1)

print("\n2 Follow-up")
res2 = agent.run("Who created it?")
print("Answer:", res2)

print("\n3 Ask again")
res3 = agent.run("Explain LangChain again simply.")
print("Answer:", res3)
```

1 First question

> Entering new AgentExecutor chain...

Thought: Do I need to use a tool? Yes

Action: RAG_QA

Action Input: What is LangChain?

Observation: LangChain is a framework created by Harrison Chase for building applications with Large Language Models (LLMs). It supports various features such as RAG, agents, memory, tools, and more. LangChain is commonly used in applications like chatbots, document Q&A, and AI workflow.

Do I need to use a tool? No

AI: LangChain is a framework created by Harrison Chase for building applications with Large Language Models (LLMs). It supports various features such as RAG, agents, memory, tools, and more. LangChain is commonly used in applications like chatbots, document Q&A, and AI workflow.

> Finished chain.

Answer: LangChain is a framework created by Harrison Chase for building applications with Large Language Models (LLMs). It supports various features such as RAG, agents, memory, tools, and more. LangChain is commonly used in applications like chatbots, document Q&A, and AI workflow.

2 Follow-up

> Entering new AgentExecutor chain...

Thought: Do I need to use a tool? No

AI: LangChain was created by Harrison Chase.

> Finished chain.

Answer: LangChain was created by Harrison Chase.

3 Ask again

> Entering new AgentExecutor chain...

Could not parse LLM output: `LangChain is a framework created by Harrison Chase for building applications with Large Language Models (LLMs). It supports features like RAG, agents, memory, tools, and more, and is commonly used in applications like chatbots, document Q&A, and AI workflow.'

For troubleshooting, visit: https://python.langchain.com/docs/troubleshooting/OUTPUT_PARSING_FAILURE

Observation: Invalid or incomplete response

Do I need to use a tool? Yes

Action: RAG_QA

Action Input: Explain LangChain simply

Observation: LangChain is a framework created by Harrison Chase for building applications using Large Language Models (LLMs). It supports various features like RAG, agents, memory, tools, and more. LangChain is commonly used in applications such as chatbots, document question and answer systems, and AI workflow management.

Do I need to use a tool? No

AI: LangChain is a framework created by Harrison Chase for building applications using Large Language Models (LLMs). It supports various features like RAG, agents, memory, tools, and more. LangChain is commonly used in application

s such as chatbots, document question and answer systems, and AI workflow management.

> Finished chain.

Answer: LangChain is a framework created by Harrison Chase for building applications using Large Language Models (LLMs). It supports various features like RAG, agents, memory, tools, and more. LangChain is commonly used in applications such as chatbots, document question and answer systems, and AI workflow management.

In []: