
🔍 LangChain Retrievers Guide

---

1. Vector Store Retriever

Description:
Uses embedding-based similarity to fetch relevant documents from a vector store like FAISS or Chroma.

Usage:
Best for semantic search, document QA, and general-purpose retrieval.

Code Example:

```
from langchain.vectorstores import FAISS
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.document_loaders import TextLoader

loader = TextLoader("data.txt")
documents = loader.load()
embedding = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(documents, embedding)
retriever = vectorstore.as_retriever()

docs = retriever.get_relevant_documents("What is LangChain?")
print(docs)
```

---

2. Self-Query Retriever

Description:
Converts natural language questions into structured queries using metadata filters and semantic search.

Usage:
Use when documents have metadata like author, date, topic.

Code Example:

```
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo
```

```
from langchain.llms import OpenAI

metadata_field_info = [
    AttributeInfo(name="author", description="Who wrote the document", type="string"),
    AttributeInfo(name="year", description="Publication year", type="integer"),
]

retriever = SelfQueryRetriever.from_llm(
    llm=OpenAI(),
    vectorstore=vectorstore,
    document_contents="Summary of document",
    metadata_field_info=metadata_field_info,
)

results = retriever.get_relevant_documents("Articles by John in 2023")
print(results)
```

---

3. Multi-Query Retriever

Description:
Creates multiple rephrasings of a query using LLMs to improve recall of relevant results.

Usage:
Best for ambiguous or broad questions where one query may miss important matches.

Code Example:

```
from langchain.retrievers.multi_query import MultiQueryRetriever
from langchain.llms import OpenAI

retriever = MultiQueryRetriever.from_llm(
    retriever=vectorstore.as_retriever(),
    llm=OpenAI()
)

results = retriever.get_relevant_documents("Explain retrievers in LangChain")
print(results)
```

---

4. Parent Document Retriever

Description:
Indexes chunks but returns full parent document when a chunk is retrieved.

Usage:
Use when full document context is necessary for understanding.

Code Example:

```
from langchain.retrievers import ParentDocumentRetriever
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.storage import InMemoryStore

splitter = RecursiveCharacterTextSplitter(chunk_size=200, chunk_overlap=20)
store = InMemoryStore()

retriever = ParentDocumentRetriever.from_components(
    text_splitter=splitter,
    vectorstore=FAISS.from_documents([], OpenAIEmbeddings()),
    docstore=store,
)

retriever.add_documents(documents)
results = retriever.get_relevant_documents("retriever architecture")
print(results)
```

---

5. Contextual Compression Retriever

Description:
Uses a compression layer (LLM or filter) to condense documents returned by a base retriever.

Usage:
Helpful when documents are long and a concise summary is more useful.

Code Example:

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainFilter
from langchain.llms import OpenAI

base_retriever = vectorstore.as_retriever()
compressor = LLMChainFilter.from_llm(OpenAI())
retriever = ContextualCompressionRetriever(base_compressor=compressor,
base_retriever=base_retriever)
```

```
results = retriever.get_relevant_documents("Explain vector retrievers briefly")
print(results)
```

---

6. Ensemble Retriever

Description:
Combines multiple retrievers (e.g., keyword + vector) and balances them using weights.

Usage:
For hybrid search scenarios combining keyword and semantic relevance.

Code Example:

```
from langchain.retrievers import EnsembleRetriever, BM25Retriever

bm25 = BM25Retriever.from_documents(documents)
bm25.k = 4
vector = vectorstore.as_retriever()

ensemble = EnsembleRetriever(retrievers=[bm25, vector], weights=[0.3, 0.7])
results = ensemble.get_relevant_documents("What are vector databases?")
print(results)
```

---

7. Time-Weighted Retriever

Description:
Applies time decay to favor recent documents during retrieval.

Usage:
Useful in chatbots, assistants, or time-sensitive document search.

Code Example:

```
from langchain.retrievers import TimeWeightedVectorStoreRetriever

retriever = TimeWeightedVectorStoreRetriever(
    vectorstore=vectorstore,
    other_score_keys=["importance"],
    decay_rate=0.01,
)
```

```
retriever.add_documents(documents)
docs = retriever.get_relevant_documents("Recent updates in LangChain?")
print(docs)
```

---

8. BM25 Retriever

Description:
Classic keyword-based retriever using BM25 scoring.

Usage:
Best for exact keyword matches.

Code Example:

```
from langchain.retrievers import BM25Retriever

retriever = BM25Retriever.from_documents(documents)
retriever.k = 5
docs = retriever.get_relevant_documents("What is LangChain?")
print(docs)
```

---

9. TF-IDF Retriever

Description:
Uses Term Frequency-Inverse Document Frequency to rank and retrieve documents.

Usage:
Good for simple, local keyword matching.

Code Example:

```
from langchain.retrievers import TFIDFRetriever

retriever = TFIDFRetriever.from_documents(documents)
docs = retriever.get_relevant_documents("LangChain modules")
print(docs)
```

---

10. ElasticSearch Retriever

Description:
Retrieves documents from ElasticSearch using keyword and full-text search.

Usage:
For enterprise-scale retrieval and structured indexing.

Code Example:

```
from langchain.retrievers import ElasticSearchBM25Retriever

retriever = ElasticSearchBM25Retriever(
    elasticsearch_url="http://localhost:9200",
    index_name="docs",
)

docs = retriever.get_relevant_documents("LLM pipelines")
print(docs)
```

---

11. Google Search Retriever (SerpAPI)

Description:
Retrieves live web results using SerpAPI (Google).

Usage:
When up-to-date web knowledge is needed.

Code Example:

```
from langchain.tools import SerpAPIWrapper

search = SerpAPIWrapper()
results = search.run("Latest LangChain updates")
print(results)
```

---

12. Web Research Retriever

Description:
Uses agent-based research with tools like search and scraping to find answers.

Usage:
Deep research or question answering over external web data.

Code Example:

```
from langchain.agents import initialize_agent, Tool
from langchain.tools import SerpAPIWrapper
from langchain.llms import OpenAI

search = SerpAPIWrapper()
tools = [Tool(name="Search", func=search.run, description="Searches the web")]
agent = initialize_agent(tools, OpenAI(), agent="zero-shot-react-description", verbose=True)

response = agent.run("Summarize new LangChain retrievers")
print(response)
```

---

13. YouTube Retriever (Custom)

Description:
Loads video transcript, splits it, stores in vector DB, and enables semantic video search.

Usage:
Perfect for building bots or search systems on video content.

Code Example:

```
from langchain.document_loaders import YoutubeLoader
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter

# 1. Load transcript from YouTube
loader = YoutubeLoader.from_youtube_url("https://www.youtube.com/watch?v=VIDEO_ID",
add_video_info=True)
docs = loader.load()

# 2. Split into chunks
splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
docs_split = splitter.split_documents(docs)

# 3. Create embeddings
embedding = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(docs_split, embedding)

# 4. Use as retriever
retriever = vectorstore.as_retriever()
```

```
results = retriever.get_relevant_documents("What is LangChain?")
for r in results:
    print(r.page_content)
```
