✅ 4. ParentDocumentRetriever

Use Case: Retrieve parent chunks based on a retrieval of smaller child chunks, great for keeping context intact in long documents.

In [ ]:
```
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.retrievers import ParentDocumentRetriever
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.storage import InMemoryStore
from langchain.schema.document import Document
from dotenv import load_dotenv
import os

# 1. Load environment
load_dotenv()
os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")
# 1. Prepare documents
docs = [Document(page_content="LangChain helps build LLM-powered apps with m
        Document(page_content="Agents in LangChain use tools to answer quest

# 2. Setup child splitter
child_splitter = RecursiveCharacterTextSplitter(chunk_size=100, chunk_overla

# 3. Setup vectorstore for children
embedding = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(docs, embedding)

# 4. Parent retriever
retriever = ParentDocumentRetriever(
    vectorstore=vectorstore,
    docstore=InMemoryStore(),  # Stores parent docs
    child_splitter=child_splitter
)

# 5. Add documents
retriever.add_documents(docs)

# 6. Retrieve
results = retriever.get_relevant_documents("What are agents?")
for doc in results:
    print("📄 Retrieved Doc:", doc.page_content)
```

✅ 5. BM25Retriever

Use Case: Pure keyword-based search (like traditional search engines). No embeddings needed.

In [5]:
```
#!pip install rank_bm25
```

```
Collecting rank_bm25
  Downloading rank_bm25-0.2.2-py3-none-any.whl.metadata (3.2 kB)
Requirement already satisfied: numpy in c:\anaconda3\envs\langchainhope\lib\s
ite-packages (from rank_bm25) (2.2.6)
Downloading rank_bm25-0.2.2-py3-none-any.whl (8.6 kB)
Installing collected packages: rank_bm25
Successfully installed rank_bm25-0.2.2
```

In [6]:
```python
from langchain.retrievers import BM25Retriever
from langchain.schema.document import Document

# Create simple text docs
docs = [
    Document(page_content="LangChain enables LLM applications."),
    Document(page_content="Vector search is powerful."),
    Document(page_content="BM25 is a classical retrieval method.")
]

# Create BM25 retriever
bm25_retriever = BM25Retriever.from_documents(docs)

# Retrieve
results = bm25_retriever.get_relevant_documents("How does BM25 work?")
for doc in results:
    print("📝 BM25 Result:", doc.page_content)
```

```
📝 BM25 Result: BM25 is a classical retrieval method.
📝 BM25 Result: Vector search is powerful.
📝 BM25 Result: LangChain enables LLM applications.
```

✅ 6. EnsembleRetriever

Use Case: Combine multiple retrievers (e.g., keyword + vector-based) with weighted scores.

In [7]:
```python
from langchain.retrievers import EnsembleRetriever
from langchain.retrievers import BM25Retriever
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.schema.document import Document

# Sample docs
docs = [Document(page_content="LangChain supports LLMs."), Document(page_con

# BM25 Retriever
bm25 = BM25Retriever.from_documents(docs)

# Vector Retriever
embedding = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(docs, embedding)
vector_retriever = vectorstore.as_retriever()

# Ensemble Retriever (equal weight)
ensemble_retriever = EnsembleRetriever(
    retrievers=[bm25, vector_retriever],
    weights=[0.5, 0.5]
)
```

```
# Query
results = ensemble_retriever.get_relevant_documents("AI apps using LangChain
for doc in results:
    print("🔍 Ensemble Doc:", doc.page_content)
```

```
🔍 Ensemble Doc: You can build AI apps using LangChain.
🔍 Ensemble Doc: LangChain supports LLMs.
```

7.TimeWeightedVectorStoreRetriever

This retriever boosts document relevance by factoring recency and importance of interactions (used in agents with memory or relevance ranking).

In [25]:
```
#7.TimeWeightedVectorStoreRetriever
#This retriever boosts document relevance by factoring recency and importanc

from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.retrievers import TimeWeightedVectorStoreRetriever
from langchain.schema import Document
from datetime import datetime
import os

# Initialize embedding & vectorstore
embedding = OpenAIEmbeddings()
docs = [
    Document(page_content="LangChain is for LLM-based apps", metadata={"last
    Document(page_content="Vector search improves relevance", metadata={"las
]
vectorstore = FAISS.from_documents(docs, embedding)

# TimeWeighted Retriever
retriever = TimeWeightedVectorStoreRetriever(
    vectorstore=vectorstore,
    decay_rate=0.01,
    k=2,
    score_threshold=None
)

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

In [24]:
```
#!pip install tavily-python
```

8.TavilySearchAPIRetriever

In [12]:
```python
from langchain.retrievers import TavilySearchAPIRetriever
import os

# Set your Tavily API key
#os.environ["TAVILY_API_KEY"] = "your_tavily_api_key"

from dotenv import load_dotenv
import os
# 🔐 Load API keys
load_dotenv(".env")
openai_api_key = os.getenv("OPENAI_API_KEY")
tavily_api_key = os.getenv("TAVILY_API_KEY")

# Initialize Tavily Retriever
retriever = TavilySearchAPIRetriever(k=3)

# Perform search
docs = retriever.get_relevant_documents("Latest updates about LangChain")
for doc in docs:
    print(doc.page_content)
```

```
LangChain v0.3. Last updated: 09.16.24. What's changed. All packages have bee
n upgraded from Pydantic 1 to Pydantic 2 internally.
Connect traces in LangSmith to server logs in LangGraph Platform · July 31, 2
025 ; Introducing Align Evals: Streamlining LLM Application Evaluation · July
29, 2025.
What's new in LangChain? · Better streaming support via the Event Streaming A
PI. · Standardized tool calling support · A standardized interface for struct
uring
```

In [ ]:

In [ ]: