Chain / Retriever	Returns Citations	Method / Output Key	Status
`RetrievalQAWithSourcesChain`	✅ Yes	`result['answer']`, `result['sources']`	**Stable**
`ConversationalRetrievalChain`	✅ Yes	`result['answer']`, `result['source_documents']`	**Stable**
`MultiRetrievalQAChain`	✅ Yes	Depends on sub-chains used	**Stable**
`VectorDBQAWithSourcesChain`	✅ Yes	`result['answer']`, `result['sources']`	✅ (but legacy)
`Tool` using RetrievalQAWithSources	✅ Yes	`result['answer']`, `result['sources']`	**Stable** (via Agent)
`RetrievalQA` (basic chain)	❌ No	Only `answer`	**Stable**
`RefineDocumentsChain`	❌ No	Only `answer`	**Stable**
`StuffDocumentsChain`	❌ No	Only `answer`	**Stable**

In [ ]:
```python
# ✅ Step 1: Imports
from langchain.chat_models import ChatOpenAI
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.text_splitter import CharacterTextSplitter
from langchain.schema import Document
from dotenv import load_dotenv
import os

# ✅ Step 2: Load API key
load_dotenv()
os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")

# ✅ Step 3: Setup LLM and Embeddings
llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")
embedding = OpenAIEmbeddings()

# ✅ Step 4: Load and split document
with open("sample.txt", "r", encoding="utf-8") as f:
    raw_text = f.read()

splitter = CharacterTextSplitter(separator="\n", chunk_size=300, chunk_overl
texts = splitter.split_text(raw_text)
documents = [Document(page_content=t) for t in texts]

# ✅ Step 5: Vector Store (optional for Retriever chains)
vectorstore = FAISS.from_texts(texts, embedding)
retriever = vectorstore.as_retriever()
```

```
print("✅ Base setup complete.")
```

In [ ]:

In [2]:
```
#✅ 4. StuffDocumentsChain
#Use case: Combines all docs into a single string before passing to LLM (bes

from langchain.chains import StuffDocumentsChain
from langchain.prompts import PromptTemplate
from langchain.chains.llm import LLMChain

# Define prompt
prompt = PromptTemplate.from_template(
    "Use the following context to answer the question:\n\n{context}\n\nQuest
)

# Inner LLM Chain
llm_chain = LLMChain(llm=llm, prompt=prompt)

# Stuff Chain
stuff_chain = StuffDocumentsChain(
    llm_chain=llm_chain,
    document_variable_name="context"
)

# Run
response = stuff_chain.invoke({
    "input_documents": documents[:3],  # test on 3 docs
    "question": "What is this document about?"
})
```

```
reated by Harrison Chase for building applications with LLMs. It discusses th
e features and uses of LangChain, such as supporting RAG, agents, memory, too
ls, and more, and its common applications in chatbots, document Q&A, and AI w
orkflow.
```

5) Runnabale

In [6]:
```python
# Initial prompt to summarize the first chunk
#Use case: Starts with a base answer and refines it using subsequent documen

from langchain.prompts import PromptTemplate
from langchain.chat_models import ChatOpenAI
from langchain_core.documents import Document
from langchain_core.runnables import Runnable
from langchain_core.output_parsers import StrOutputParser

initial_prompt = PromptTemplate.from_template("""
Write a concise summary of the following text:

{context}
""")

# Refine prompt to update the previous summary with new context
refine_prompt = PromptTemplate.from_template("""
We have an existing summary:
"{existing_answer}"

Refine the summary with this new context:
"{context}"

If the context isn't useful, return the original summary.
""")

# Set up individual chains
initial_summary_chain = initial_prompt | llm | StrOutputParser()
refine_summary_chain = refine_prompt | llm | StrOutputParser()

# Start with first chunk
summary = initial_summary_chain.invoke({"context": documents[0].page_content

# Iteratively refine with remaining docs
for doc in documents[1:]:
    summary = refine_summary_chain.invoke({
        "existing_answer": summary,
        "context": doc.page_content
    })

# Output final summary
print("📄 Refined Summary:\n")
print(summary)
```

```
📄 Refined Summary:

LangChain is a framework developed by Harrison Chase for building application
s with LLMs. It supports various features such as RAG, agents, memory, and to
ols, and is commonly used in chatbots, document Q&A, and AI workflow.
```

In [ ]:
```
```