In [2]:
```python
# 1. Imports
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain_core.documents import Document
from langchain_core.runnables import RunnableParallel, RunnableLambda
from langchain.text_splitter import CharacterTextSplitter

import os
from dotenv import load_dotenv
load_dotenv()

# 2. Setup
llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")

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

splitter = CharacterTextSplitter(separator="\n", chunk_size=300, chunk_overl
docs = splitter.create_documents([raw_text])

# 4. Define prompts
map_prompt = PromptTemplate.from_template("Summarize:\n\n{context}")
reduce_prompt = PromptTemplate.from_template("Combine the following summarie

# 5. Create map + reduce chains
map_chain = map_prompt | llm
reduce_chain = reduce_prompt | llm

# 6. Use RunnableLambda to convert documents
def map_docs(docs):
    return [{"context": doc.page_content} for doc in docs]

def extract_texts(responses):
    return {"context": "\n".join([res.content for res in responses])}

# 7. Combine using RunnableParallel (MapReduce style)
map_reduce_chain = (
    RunnableLambda(map_docs)
    | map_chain.map()
    | RunnableLambda(extract_texts)
    | reduce_chain
)

# 8. Execute
result = map_reduce_chain.invoke(docs)
print("📄 Final Summary:\n", result.content)
```

```
📄 Final Summary:
 LangChain is a versatile framework developed by Harrison Chase for construct
ing applications using LLMs. It includes support for RAG, agents, memory, too
ls, and other features, making it a popular choice for applications such as c
hatbots, document Q&A, and AI workflow.
```

```
In [ ]:
```