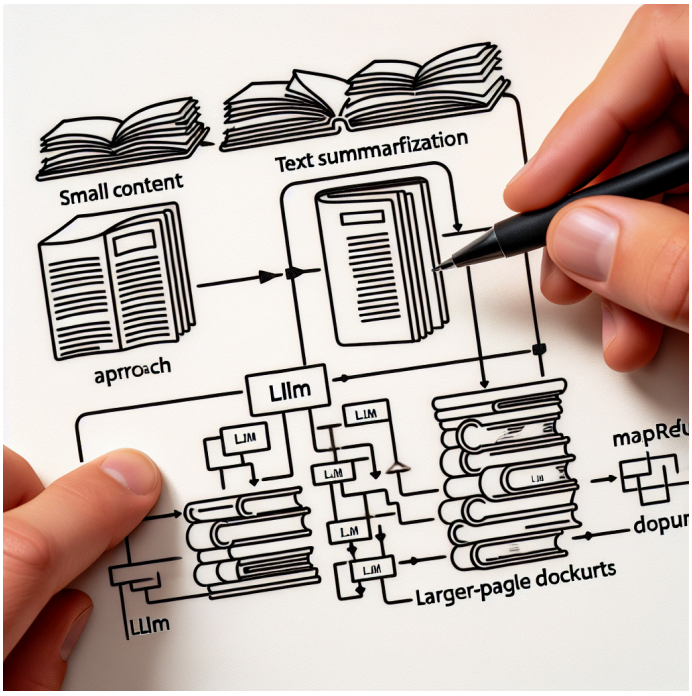
Summarization with GenAI: Approach to Summarizing Text Documents



Continuing with last week's [article], this week we will look at different approaches to summarizing text documents.

The crux of the problem for summarizing text content using LLM is: Send the content to the LLM along with the prompt, which is the input. The LLM will give us back the summarized content, which is the output.

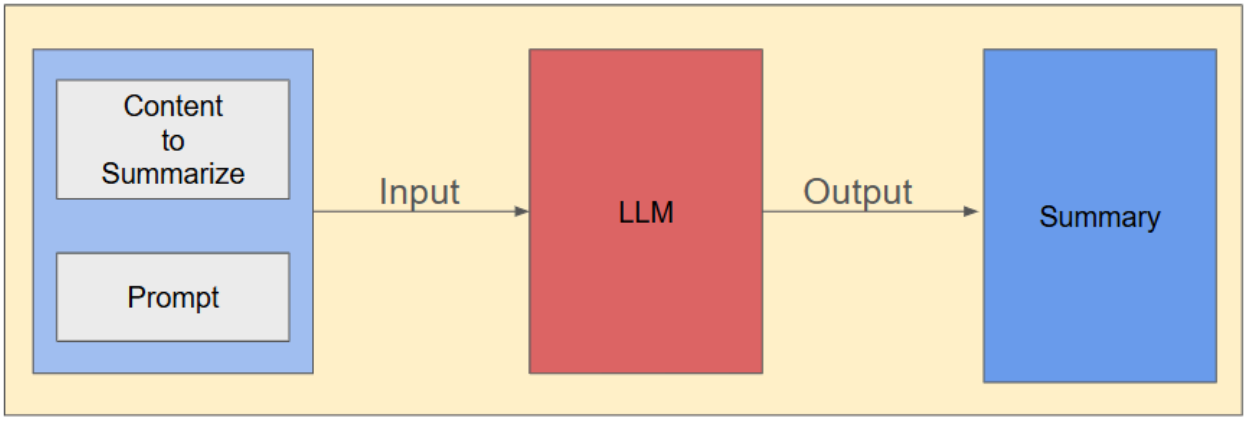


Fig :1 Text Summarizing

The questions we have to address here are:

1. Is the content to be summarized short text or long text?
2. How do I create an effective prompt?

Let us address the first question.

1. Is the Content to Summarize Short Text or Long Text?

First, we have to understand what short text is. Is it one paragraph, one page, five pages, or anything else? This is generally answered like this: the text which is longer than the token limit or context window of the LLM is long text. This is different for different models. For example:

- GPT-4 has a large context window with 128,000 tokens.
- GPT-3.5-turbo has a smaller context window with 4,096 tokens.

Note: A helpful rule of thumb is 100 tokens ≈ 75 words.

Summarizing Short Text

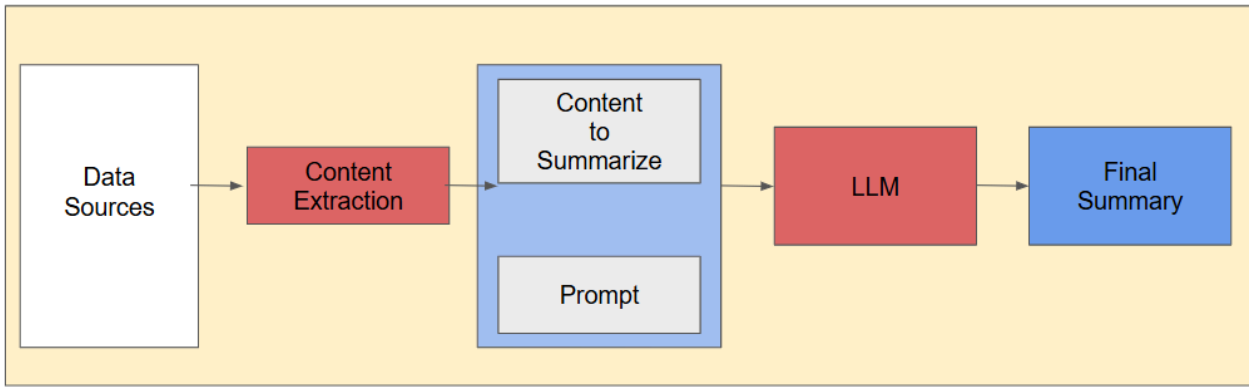
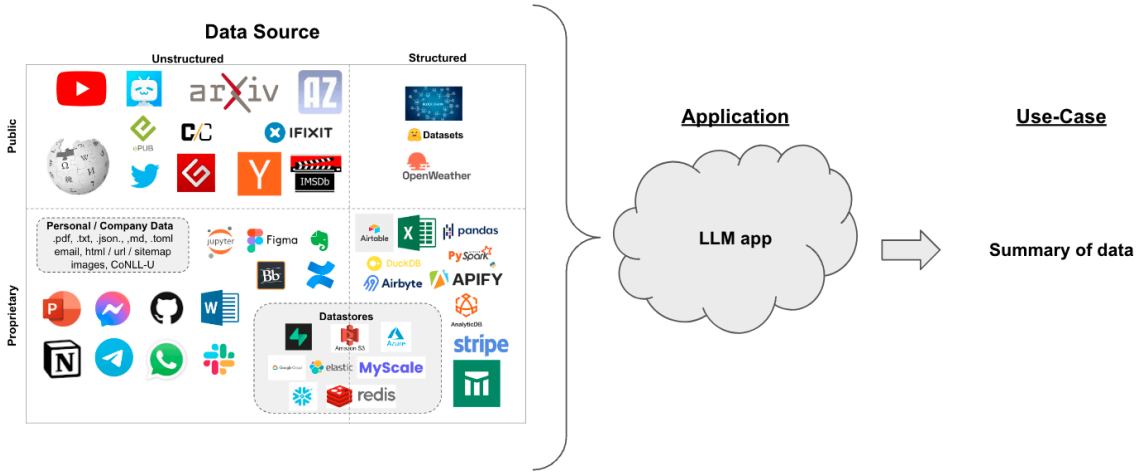


Fig 2: Text Summarizing - Short Text

The steps involved here are:

- Extract the content from the data sources. The data sources can vary widely, from files, data from different data sources, data from APIs, etc. A good representation is below:



Ref:https://python.langchain.com/v0.1/assets/images/summarization_use_case_1-874f7b2c94f64216f1f967fb5aca7bc1.png

- Prepare the prompt along with the content.
- Send this to the LLM.
- The result we get from the LLM is the final summary.

A sample code

```python
#1.Extract the content from the data sources.
loader = PyPDFLoader("content.pdf")
docs = loader.load()

#2.Prepare the Prompt along with the content.
prompt_template = """Write a concise summary of the following:
"{text}"
CONCISE SUMMARY:"""
prompt = PromptTemplate.from_template(prompt_template)
print(docs)

#3.Prepare the LLM and send this to LLM.
llm = ChatOpenAI(api_key= API_KEY,temperature=0, model_name="gpt-3.5-turbo-16k")
llm_chain = LLMChain(llm=llm, prompt=prompt)
stuff_chain = StuffDocumentsChain(llm_chain=llm_chain, document_variable_name="text")

#4. The result we will get from the LLM is the final summary.
finalSummary=stuff_chain.run(docs)
print(finalSummary)
```

This code uses the Langchain framework. Most of the code is from Summarization | 🦜🔗 LangChain

Summarizing Long Text

Summarizing Long Text

Summarizing long text follows similar steps with slight variations. This approach/strategy is also called the summary-of-summaries approach.

The technique would be to break the documents into smaller documents (chunks) which fits the context window and generate a summary for each chunk.Then combine all these chunk summaries to generate a final summary.

There are multiple approaches to chunk the documents depending on the business needs. Let us see a simple approach using Character Text Splitter with MapReduce.

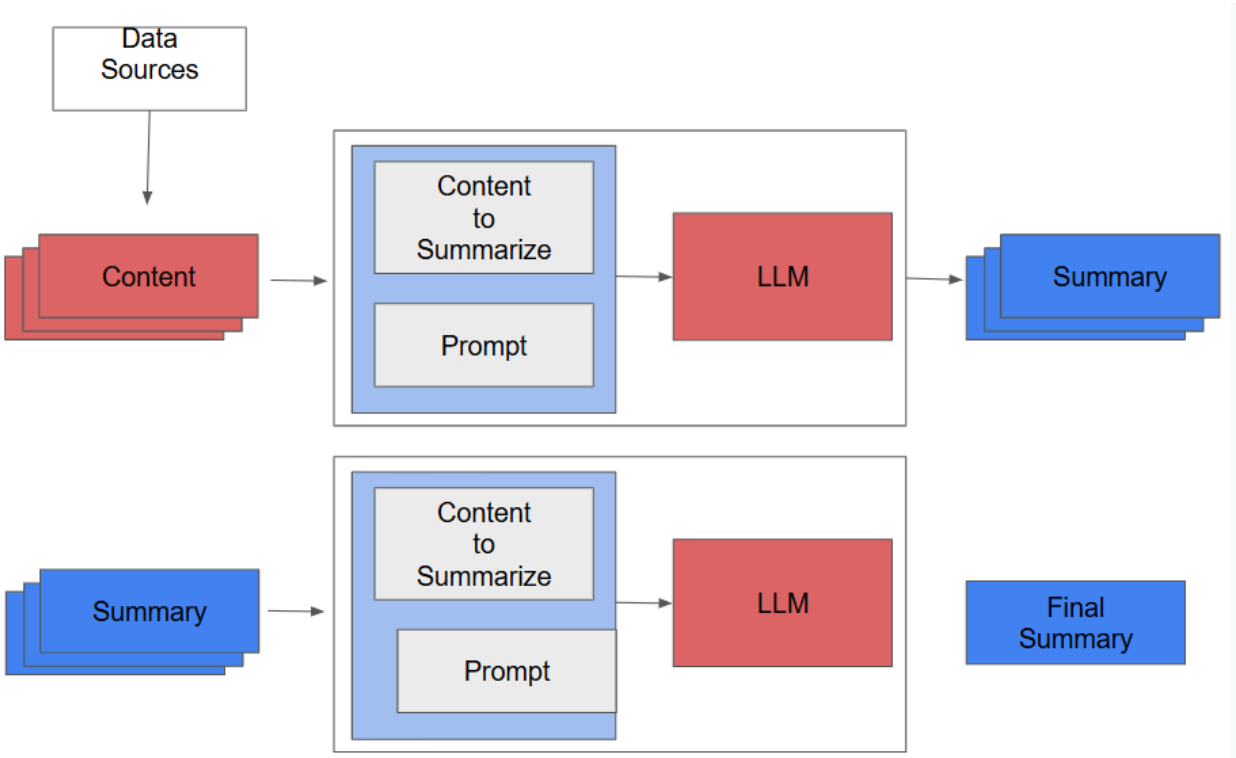


Fig 3: Text Summarizing - Long Text

In this there are 3 Main Steps:

1. **Load and split the document**: The first step is the Map Step, where the original content is split into smaller chunks of content that can fit into an LLM. We can use different techniques to split the whole content into small chunks.

```
# Load and split the document
loader = PyPDFLoader("AI102SDK.pdf")
docs = loader.load()
print("Initializing CharacterSplitter------------------")
text_splitter = CharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=1000, chunk_overlap=0
)
print(docs)
split_docs = text_splitter.split_documents(docs)
```

Ref: Summarization | 🦜🔗 LangChain

2. **Summarize each chunk**: Each of these chunks is sent to the LLM to create a summary for them (creating a summary for each chunk of content).

```
# Map
map_template = """The following is a set of documents
{docs}
Based on this list of docs, please identify the main themes
Helpful Answer:"""
map_prompt = PromptTemplate.from_template(map_template)
map_chain = LLMChain(llm=llm, prompt=map_prompt)

mappedDocs=map_chain.run(split_docs)
print(mappedDocs)
print("-------map chain ends------------------------")
```

Ref: Summarization | 🦜🔗 LangChain

3. **Combine summaries**: The next step is the Reduce Step, where we combine all the summaries from the first step and pass this to the LLM to create the final summary (this is the Summary of Summaries).

```
# Reduce
reduce_template = """The following is set of summaries:
{docs}
Take these and distill it into a final, consolidated summary of the main themes with atleast 5 bullet points.
Helpful Answer:"""
reduce_prompt = PromptTemplate.from_template(reduce_template)
reduce_chain = LLMChain(llm=llm, prompt=reduce_prompt)
# Takes a list of documents, combines them into a single string, and passes this to an LLMChain
combine_documents_chain = StuffDocumentsChain(
    llm_chain=reduce_chain, document_variable_name="docs"
)
# Combines and iteratively reduces the mapped documents
reduce_documents_chain = ReduceDocumentsChain(
    # This is final chain that is called.
    combine_documents_chain=combine_documents_chain,
    # If documents exceed context for `StuffDocumentsChain`
    collapse_documents_chain=combine_documents_chain,
    # The maximum number of tokens to group documents into.
    token_max=4000,
)
# Run chain
print(reduce_chain.run(mappedDocs))
```

Ref: Summarization | 🦜🔗 LangChain

Factors to Consider

1. The above summarisation runs an abstractive summarization in both the steps.
2. There are alternate approaches like using a combination of extractive and abstractive summarization.
3. There are different content splitting techniques which can be used. For example, if we have to summarize a book, we can split it based on chapters and summarize each chapter and then create an overall summary from that.This may be a more effective approach than simple text splitting.
4. Another option is to identify the key topics in the document and summarize along the key topics.
5. The selection of models in Step 2 can also vary; we can use specialized models for summarization like BART or PEGASUS, which are specifically trained for summarization.
6. Readers may want to explore other techniques like Map ReRank, which focuses on re-ranking the content based on relevance before summarization, to improve the quality of the summary.

So far, we have answered the first question: Is the content to summarize short text or long text, and how do we handle this?

2. How Do I Create an Effective Prompt?

So far we have answered the first question: Is the content to summarize short text or long text and how to handle this.

The next step is prompt engineering, focusing on:

- Creating a persona or role for the model for summarization.
- Giving clear context: if the content to be summarized is known, specifically mention the points that will be covered.
- Giving a clear objective of the summarization task.
- Specifying the length of the summary that is expected.

Next Week: Let us explore Text with Images

Reference:

Summarization | 🦜🔗 LangChain
Techniques for automatic summarization of documents using language models | AWS Machine Learning Blog (amazon.com)
Prompt engineering for foundation models - Amazon SageMaker
Query-based document summarization - Azure Architecture Center | Microsoft Learn
GenAI - Document Summarization using LangChain Framework | by Srinidhi G | Medium | Medium
Large Language Models and Text Summarization: A Powerful Combination | by rajni singh | Medium