Course Notes

Introduction

Prompting an LLM allows to develop AI application much faster

LLM requires prompting and parsing and more, and LangChain provides a framework with modular components to address building LLM application.

Lang Chain started as an opensource framework to address building complexity in application

Community adoption is thrilling!

Numerous users and many hundreds of contributors

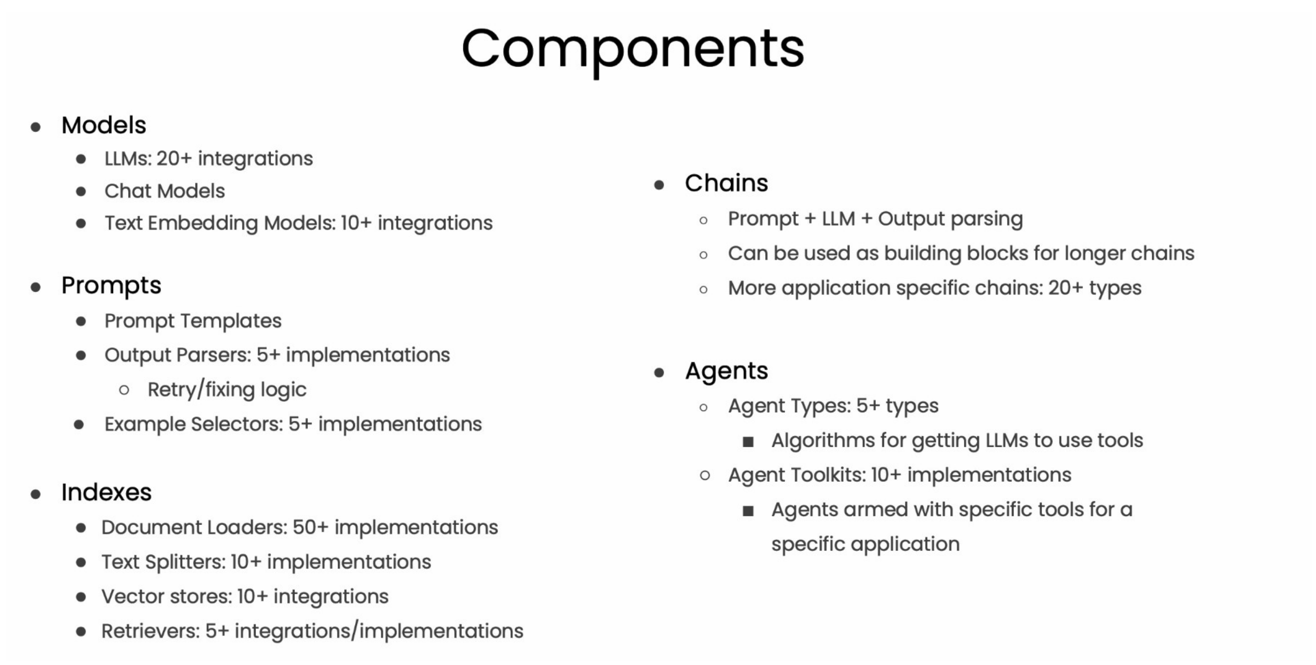
Shipping code and feature at an amazing path!!

Welcome all contribution 😊

Two packages: Python and JavaScript (TypeScript)

Key value:

- Modular components
- Uses case: common way to combine components



Components

- Models
  - LLMs: 20+ integrations
  - Chat Models
  - Text Embedding Models: 10+ integrations
- Prompts
  - Prompt Templates
  - Output Parsers: 5+ implementations
    - Retry/fixing logic
  - Example Selectors: 5+ implementations
- Indexes
  - Document Loaders: 50+ implementations
  - Text Splitters: 10+ implementations
  - Vector stores: 10+ integrations
  - Retrievers: 5+ integrations/implementations
- Chains
  - Prompt + LLM + Output parsing
  - Can be used as building blocks for longer chains
  - More application specific chains: 20+ types
- Agents
  - Agent Types: 5+ types
    - Algorithms for getting LLMs to use tools
  - Agent Toolkits: 10+ implementations
    - Agents armed with specific tools for a specific application

Models, prompts and Parsers

Models: language models (mistral, ChatGPT, etc.)

Prompts: the style of creating input to pass in to the models

Parser: taking the output and parsing it so it can be usable for further uses

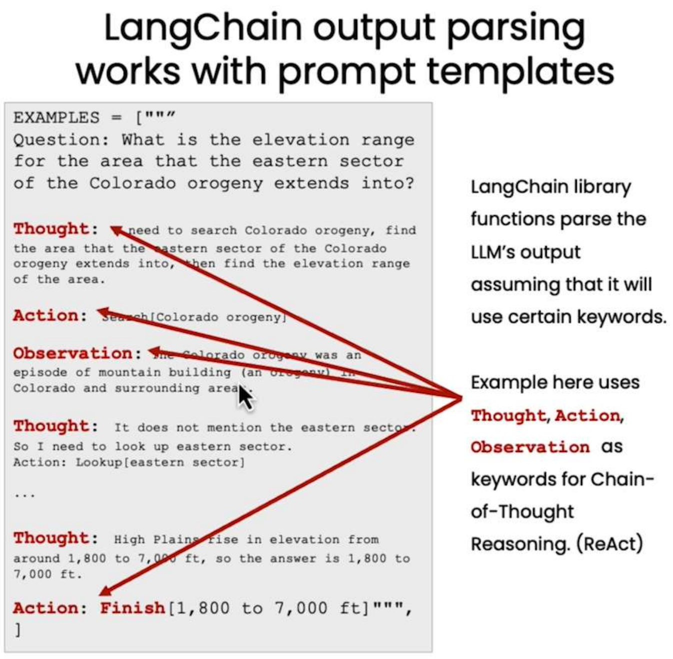
ChatPromptTemplate

Create from an f string into a function that can format message into that prompt given inputs. The prompt can then be passed to the LLM.

Neat way to write clearer and reusable prompt which can be long and detailed.

LangChain provide prompt templates for common operation (summarization, QA, connecting to SQL db, etc..).

Also help to create LLM prompt reasoning and help the parsing of the output:



LangChain output parsing works with prompt templates

```
EXAMPLES = ["""
Question: What is the elevation range
for the area that the eastern sector
of the Colorado orogeny extends into?

Thought: I need to search Colorado orogeny, find
the area that the eastern sector of the Colorado
orogeny extends into, then find the elevation range
of the area.

Action: Search[Colorado orogeny]

Observation: The Colorado orogeny was an
episode of mountain building (an orogeny) in
Colorado and surrounding area.

Thought: It does not mention the eastern sector.
So I need to look up eastern sector.
Action: Lookup[eastern sector]

...

Thought: High Plains rise in elevation from
around 1,800 to 7,000 ft, so the answer is 1,800 to
7,000 ft.

Action: Finish[1,800 to 7,000 ft]""",
]
```

LangChain library functions parse the LLM's output assuming that it will use certain keywords.

Example here uses **Thought**, **Action**, **Observation** as keywords for Chain-of-Thought Reasoning. (ReAct)

ResponseSchema

Create a response schema

StructuredOutputParser

Use the response schema to create a dictionnary out of the output

Memory

Model don’t remember what is said previously.

LangChain has multiple sophisticated option to address this issue.

ConversationBufferMemory

Used for storing previous exchanges like in a chatbot

Memory has methods and attributes like save_context(), load_memory_variable(), buffer, etc..

ConversationChain

Useful chain to create a chatbot using memory

ConversationBufferWindowsMemory

Similar to ConversationBufferMemory, but with a window that limits the amount of previous exchange stored.

➔ k: number of exchanges (= a pair of Human + Ai message)

ConversationTokenBufferMemory

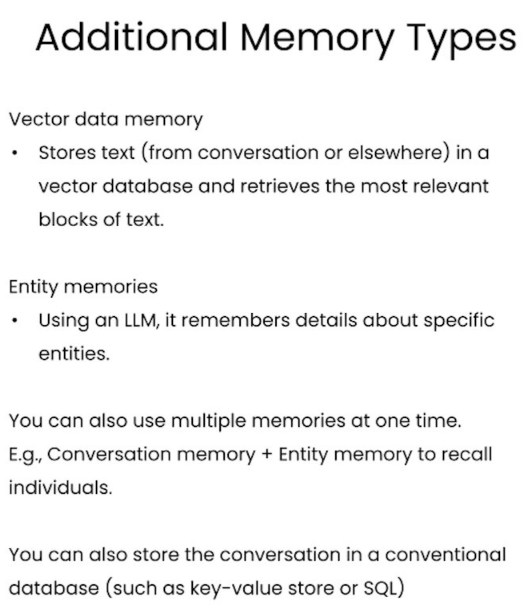
Similar to ConversationBufferMemory, but with a number of tokens that limits the amount of previous exchange stored.

➔ Max_token_limit: number of tokens to recall in the conversation.

ConversationTokenBufferMemory

Instead of limiting the number of token or exchange, an LLM write a summary of the conversation and that gives the history of the conversation

➔ Max_token_limit: number of tokens for the summary



Additional Memory Types

Vector data memory

- Stores text (from conversation or elsewhere) in a vector database and retrieves the most relevant blocks of text.

Entity memories

- Using an LLM, it remembers details about specific entities.

You can also use multiple memories at one time. E.g., Conversation memory + Entity memory to recall individuals.

You can also store the conversation in a conventional database (such as key-value store or SQL)

Chains

A chain combine an LLM with a prompt and more !

Can be combine with prompt template

Chains can be run over many inputs at a time

LLMChain

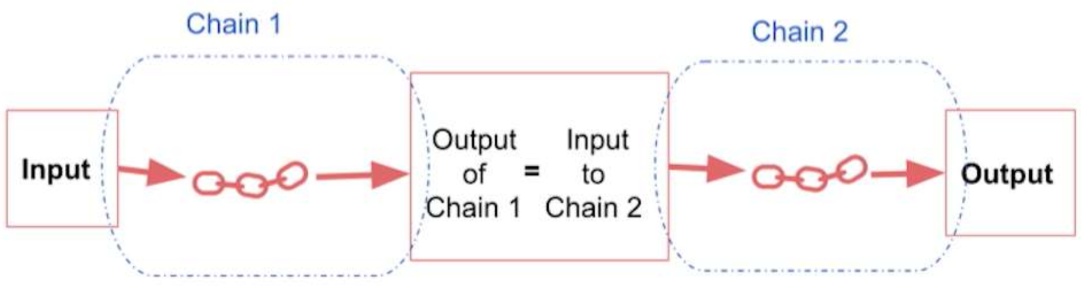
Basic building bloc: A simple chain taking a prompt and a llm.

Inputs from the prompt template becomes input for the Chain

SimpleSequentialChain

Run a sequence of chains in a sequence

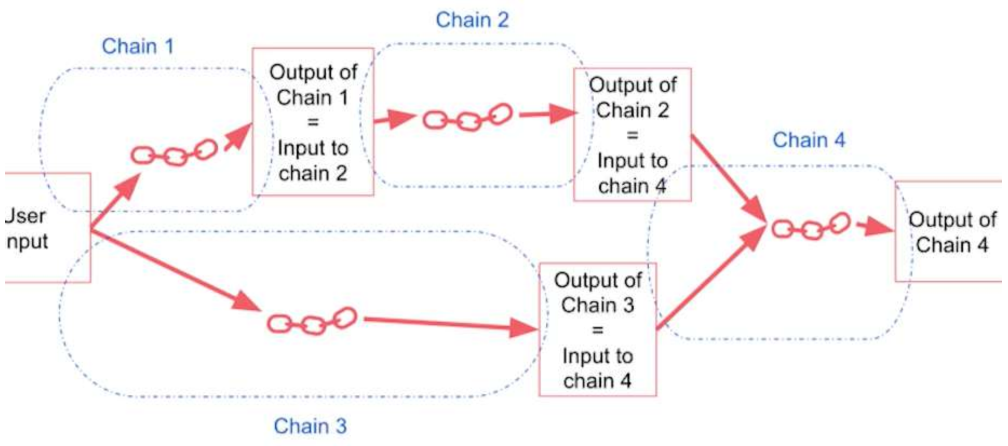
Good when there is only single input and single output between chains



SimpleSequentialChain

Run a sequence of chains in a sequence

Good when there are multiple inputs and outputs. Allow to specify inputs and output of each chains in the sequences, an input that can be an output of a previous chain

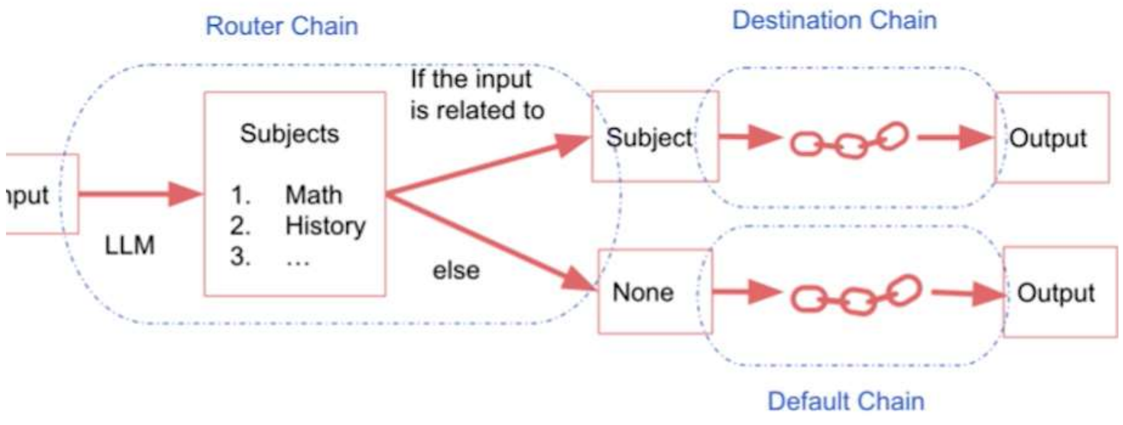

Chain 1
Chain 2
Output of Chain 1 = Input to chain 2
Output of Chain 2 = Input to chain 4
Chain 4
Jser nput
Output of Chain 4
Output of Chain 3 = Input to chain 4
Chain 3

Router Chain

Decide which chain to go next according to the prompt info.(Eg, Mixture of Expert)

Combine the following: MultiPromptChain, LLMRouterChain and RouterOutputParser

Can take a default chain when not good option is available regarding the input



Question and Answer

Allow to feed data to a model, and use the LLM as an engine to retrieve information from documents

Embedding and vector store are most powerful modern technics

This uses more complex chain already developed and available in LangChain.

RetrievalQA (Chain)

The common chain for Question Answering

Take a chain type: stuff (send all chunk) map_reduce, refine, map_rerank (experimental)

VectorstoreIndexCreator (Document loader)

Create a vector store

DocArrayInMemorySearch (vectorstores)

A vectorstore that can be use in VectorstoreIndexCreator

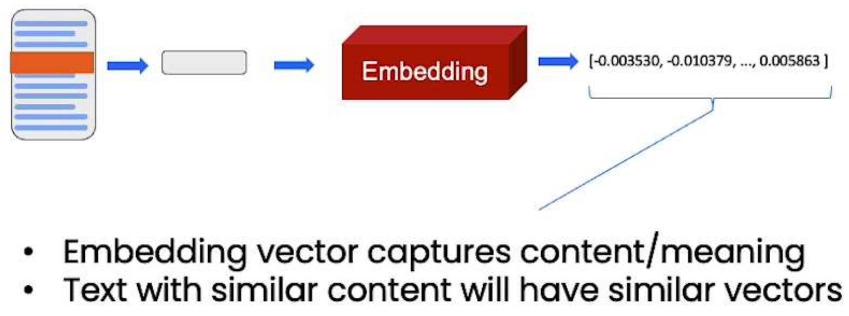
Has a method from_document() to load from documentloader

Also allow similarity search.

Embedding

Allow to store data in a more lightweight way and also to perform operation like similarity search

Similar vector means similar meaning in the original text.

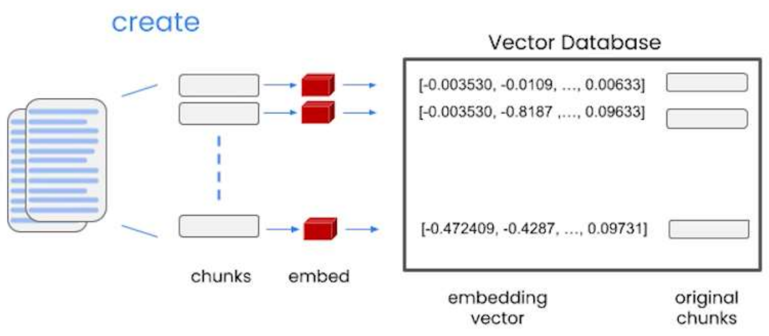


- Embedding vector captures content/meaning
- Text with similar content will have similar vectors

Vector Database

Store embedding vector as an index and the original chunk of text / documents

Allow to retrieve chunk very efficiently from index.



Retriever

Many types of retrievers: most similar, MMR, LLM aided retriever.

Evaluation

How to evaluate an LLM application?

Before all, create a chain to evaluate.

First method:

come up with good relevant chunk of text, find question and answer and create a pair by hand. Does not scale well

Second Method

QAGenerateChain with its method apply_and_parse that will generate a bunch of (question; answer) pairs

To check more on whats happen in the whole process between the input and the output of the chain, one ca use the util langChain.debug = True. There is also LangSmith that provides a nice interface.

QAEvalChain evaluate the chain given the of (question ; answer) pairs created by QAGenerateChain and return the number of correctly answer question (can also return partially correct or false).

The evaluation is done by an LLM model that compare the response of the chain to evaluate (predicted answer) with the response in the set (ground truth answer)

Agents

LLM are more than a knowledge source, it's a reasoning engine

Use the information given to decide what to do next.

Newer and most interesting part in LangChain

Agent are specific operator designed to do a specific task. They can call API (Wikipedia, YouTube, etc..), run code or chat model and more.

Mult agent chain can solve complex tasks by working together

Many agents are available on LangChain, but one can develops its own.

Sometime agents act a little funny.

Something exciting to dive in !!

Conclusion

In this short course, dwelling into a range of applications

Showing how with reasonable amounts of lines of code allows to build LLM application

LLM are really powerful because that can be used for many different tasks

LangChain Community is great 😊

Run pip install LangChain !!