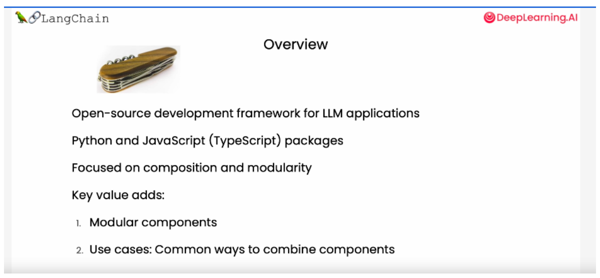
LangChain for LLM Application Development

https://learn.deeplearning.ai/langchain/lesson/1/introduction

Intro



LangChain

DeepLearning.AI

Overview

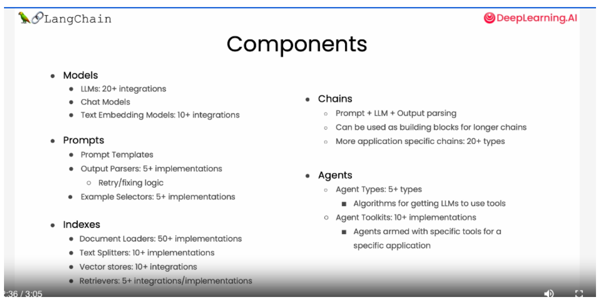
Open-source development framework for LLM applications

Python and JavaScript (TypeScript) packages

Focused on composition and modularity

Key value adds:

1. Modular components
2. Use cases: Common ways to combine components



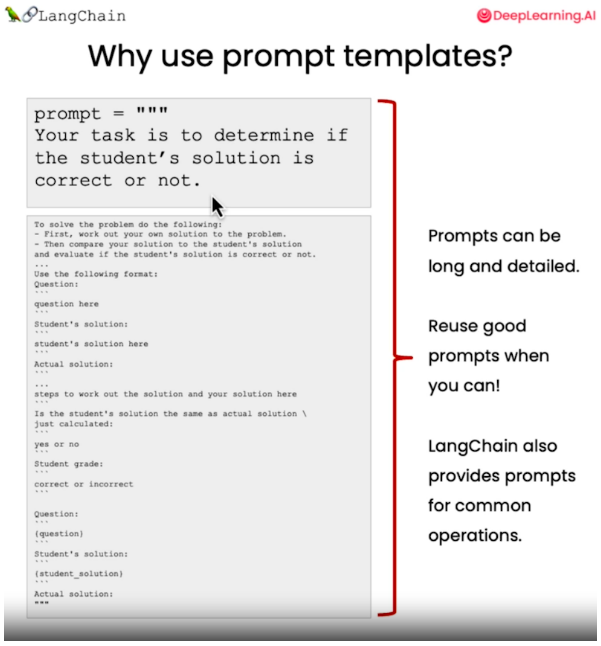
LangChain

DeepLearning.AI

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 Output Parsers



Why use prompt templates?

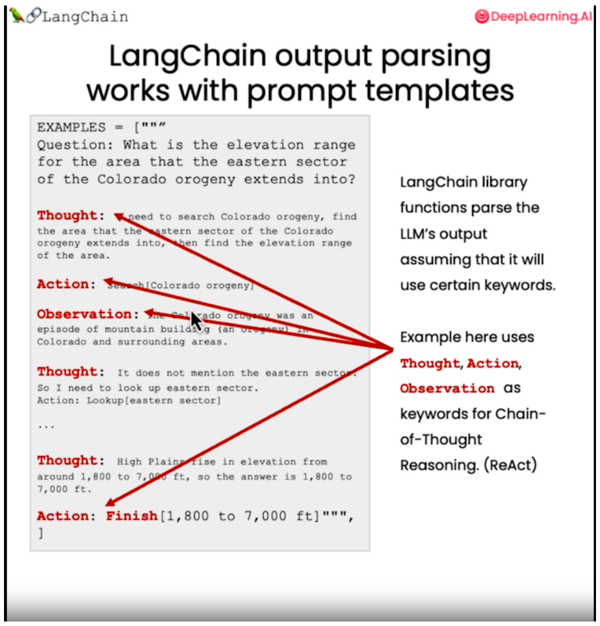
```
prompt = """
Your task is to determine if
the student's solution is
correct or not.
```

```
To solve the problem do the following:
- First, work out your own solution to the problem.
- Then compare your solution to the student's solution
and evaluate if the student's solution is correct or not.
...
Use the following format:
Question:
```
question here
```
Student's solution:
```
student's solution here
```
Actual solution:
```
...
steps to work out the solution and your solution here
```
Is the student's solution the same as actual solution \
just calculated:
```
yes or no
```
Student grade:
```
correct or incorrect
```

Question:
```
{question}
```
Student's solution:
```
{student_solution}
```
Actual solution:
"""
```

Prompts can be long and detailed.

Reuse good prompts when you can!

LangChain also provides prompts for common operations.



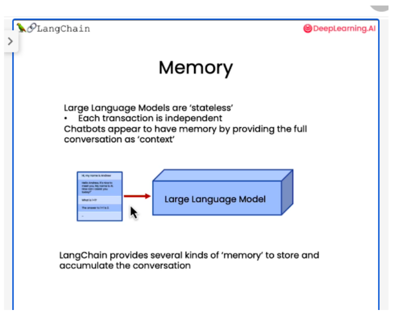
LangChain output parsing works with prompt templates

```
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 areas.

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)

Memory

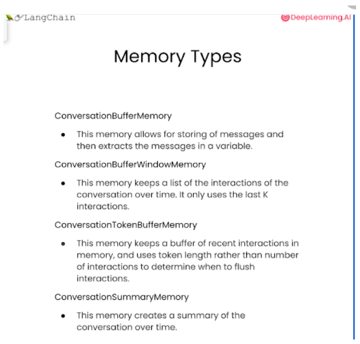


Memory

Large Language Models are 'stateless'
- Each transaction is independent

Chatbots appear to have memory by providing the full conversation as 'context'

LangChain provides several kinds of 'memory' to store and accumulate the conversation



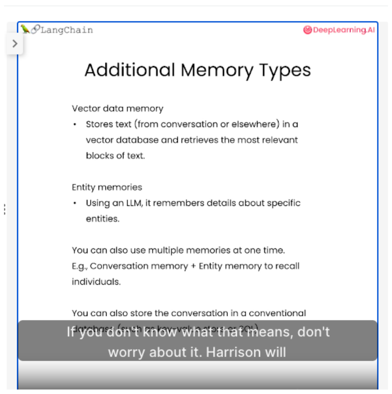
Memory Types

ConversationBufferMemory
- This memory allows for storing of messages and then extracts the messages in a variable.

ConversationBufferWindowMemory
- This memory keeps a list of the interactions of the conversation over time. It only uses the last K interactions.

ConversationTokenBufferMemory
- This memory keeps a buffer of recent interactions in memory, and uses token length rather than number of interactions to determine when to flush interactions.

ConversationSummaryMemory
- This memory creates a summary of the conversation over time.



Additional Memory Types

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

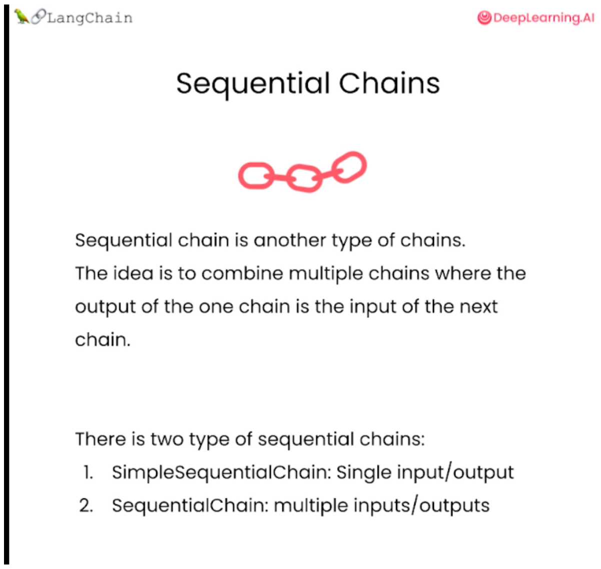
Entity memories
- Using an LLM, it remembers details about specific entities.

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)

If you don't know what that means, don't worry about it. Harrison will

Chain



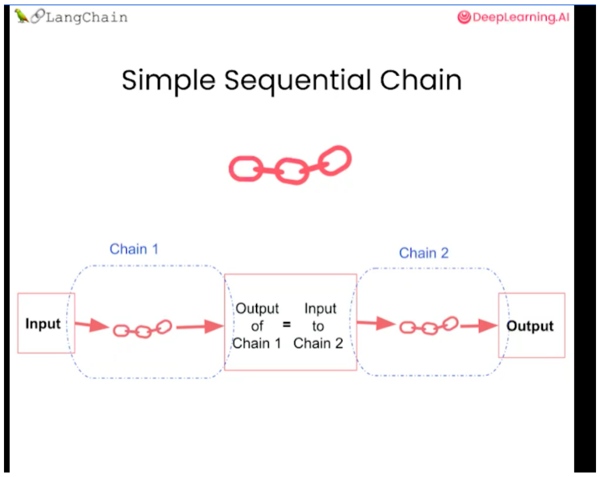
LangChain DeepLearning.AI

Sequential Chains

Sequential chain is another type of chains.
The idea is to combine multiple chains where the output of the one chain is the input of the next chain.

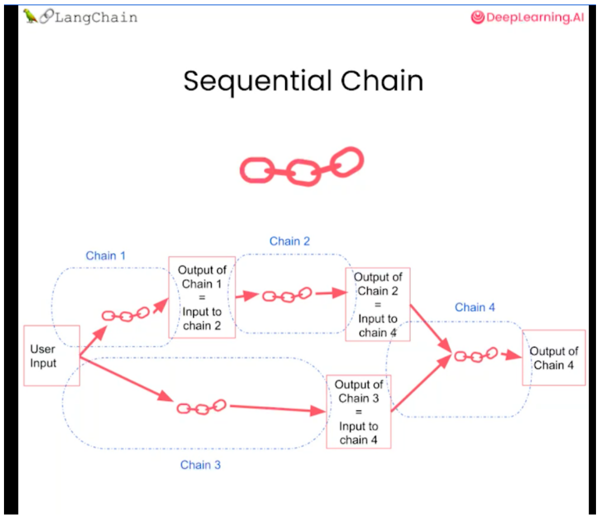
There is two type of sequential chains:

1. SimpleSequentialChain: Single input/output
2. SequentialChain: multiple inputs/outputs



LangChain DeepLearning.AI

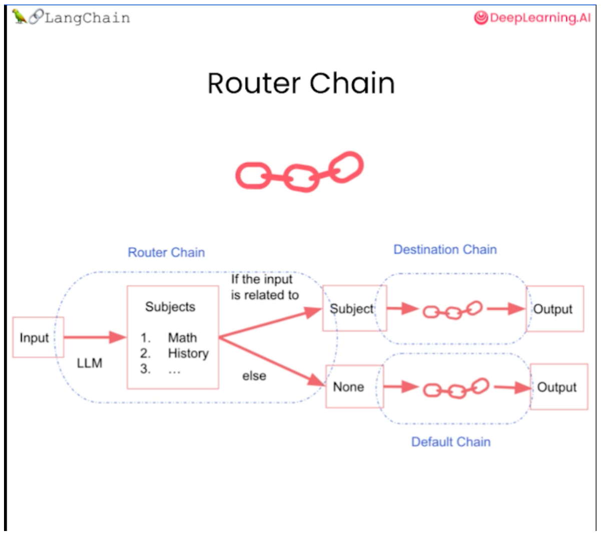
Simple Sequential Chain



LangChain DeepLearning.AI

Sequential Chain

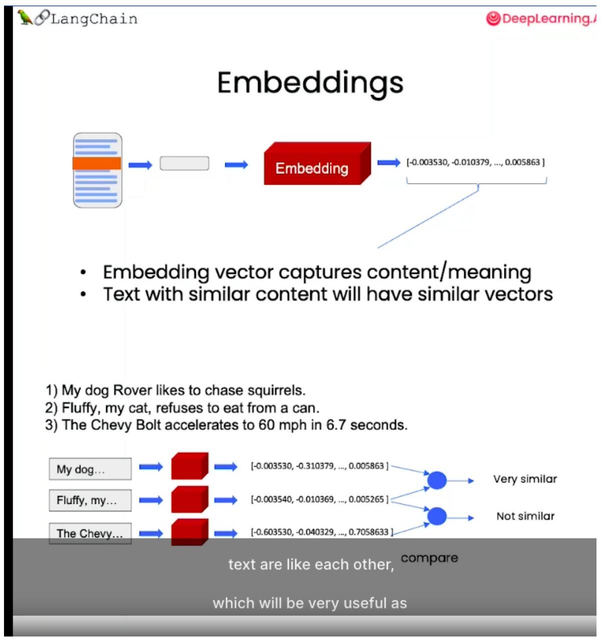
Chain 1
Chain 2
Chain 3
Chain 4
User Input
Output of Chain 1 = Input to chain 2
Output of Chain 2 = Input to chain 4
Output of Chain 3 = Input to chain 4
Output of Chain 4



LangChain DeepLearning.AI

Router Chain

Question and Answer



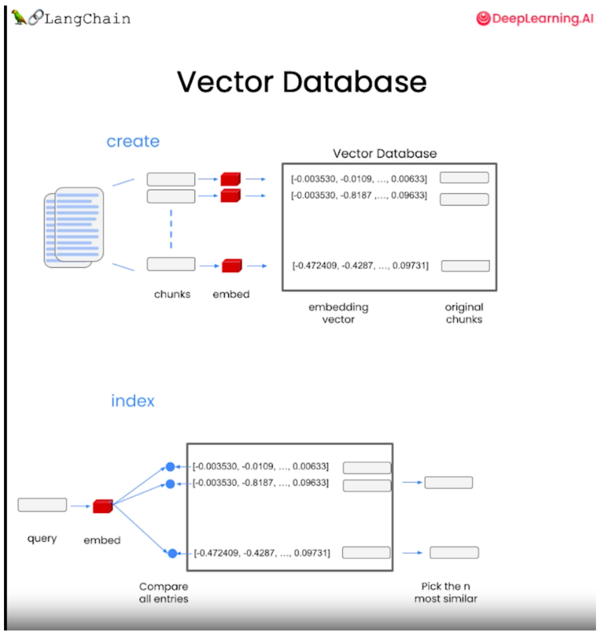
Embeddings

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

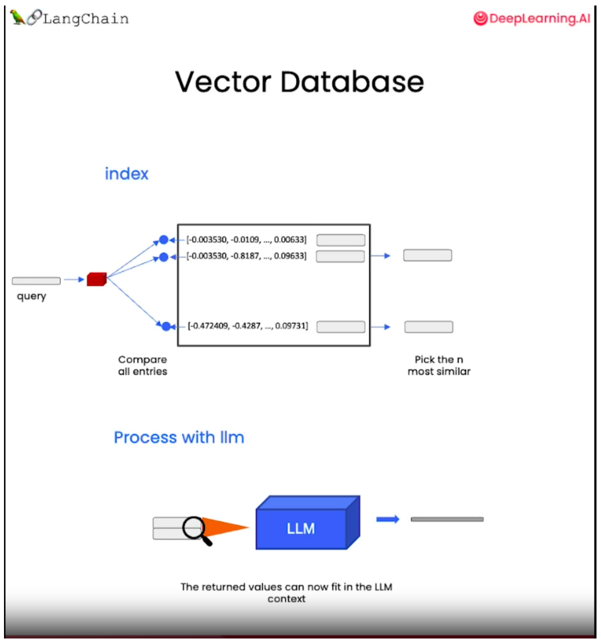
1) My dog Rover likes to chase squirrels.
2) Fluffy, my cat, refuses to eat from a can.
3) The Chevy Bolt accelerates to 60 mph in 6.7 seconds.

text are like each other,

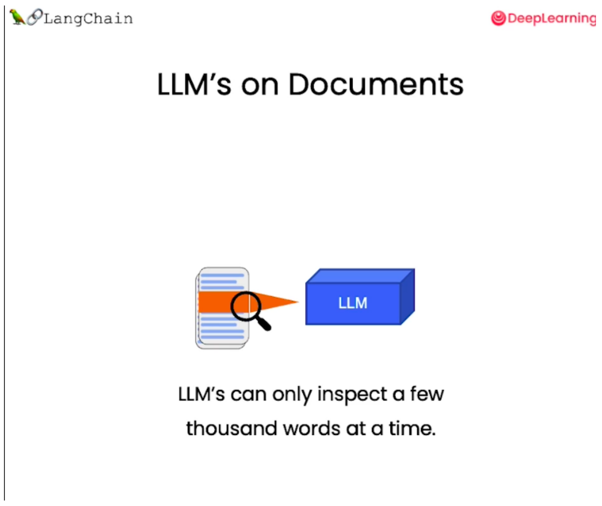
which will be very useful as



Vector Database

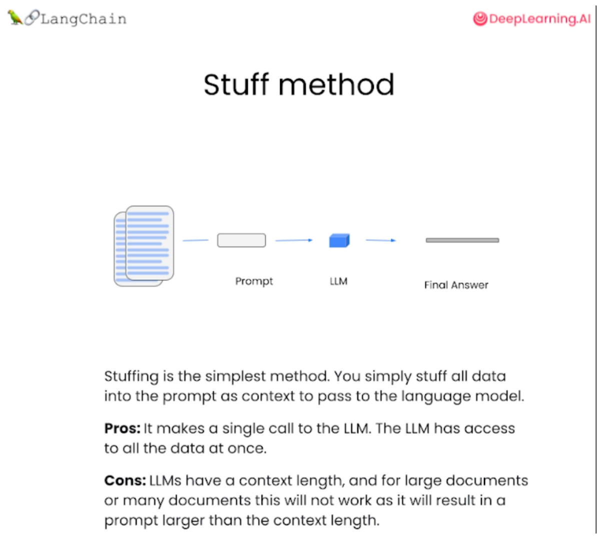


Vector Database



LLM's on Documents

LLM's can only inspect a few thousand words at a time.

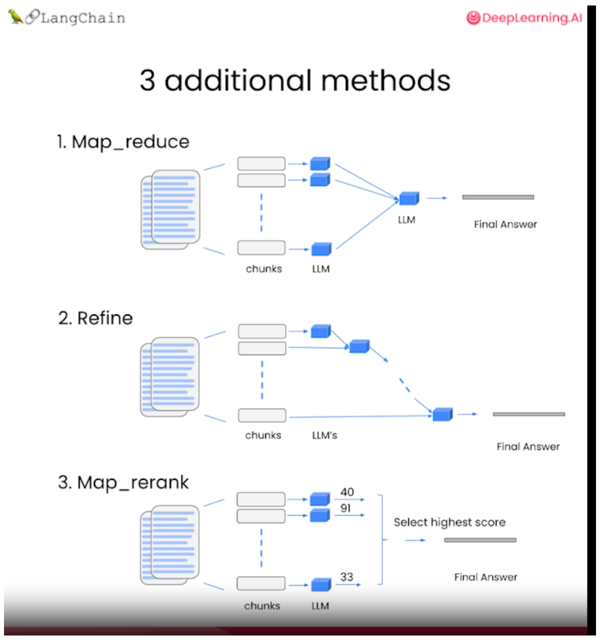

LangChain
DeepLearning.AI
Prompt
LLM
Final Answer

Stuff method

Stuffing is the simplest method. You simply stuff all data into the prompt as context to pass to the language model.

Pros: It makes a single call to the LLM. The LLM has access to all the data at once.

Cons: LLMs have a context length, and for large documents or many documents this will not work as it will result in a prompt larger than the context length.



3 additional methods

Evaluation

Agents