LangChain for LLM Application Development

Overview

- Open-source development framework for LLM applications
- Python and Javascript (TypeScript) packages
- Focused on composition and modularity
- Key value adds:
  - Modular components
  - Use cases: Common ways to combine components

Components

Models

- LLMs: 20+ integrations
- Chat models
- Text embedding models: 10+ integrations

Prompts

- Prompt templates
- Output parsers: 5+ implementations
- Example selectors: 5+ implementations

Indexes

- Document loaders: 50+ implementations
- Text Splitters: 10+ implementations
- Vector stores: 10+ implementations
- Retrievers: 5+ implementations/integrations

Chains

- Prompt + LLM + Output parsing
- Can be used as a building blocks for longer chains
- More application specific chains: 20+ types

Agents

- Agents types: 5+ types
  - Algorithms for getting LLMs to use tools
- Agent toolkits: 10+ implementations
  - Agents armed with specific tools for a specific applications

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]""",
]
```

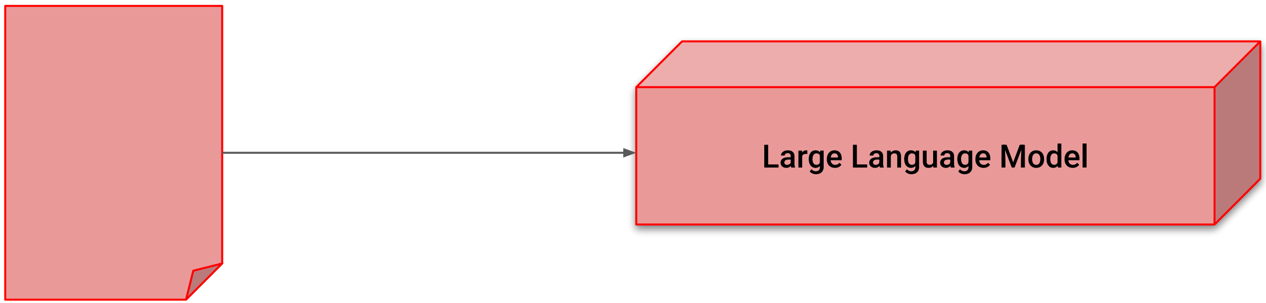
LangChain library functions parse the LLM's output assuming that it will use certain keywords.

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

Memory

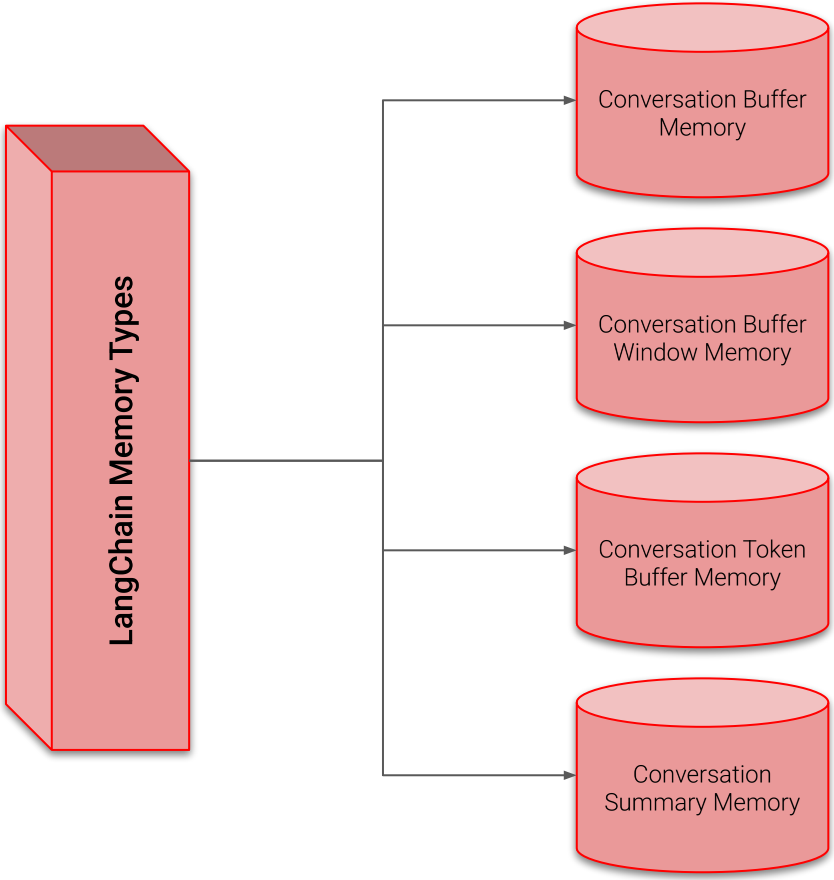
- Large language models are ‘stateless’
  - Each transaction is independent

- 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



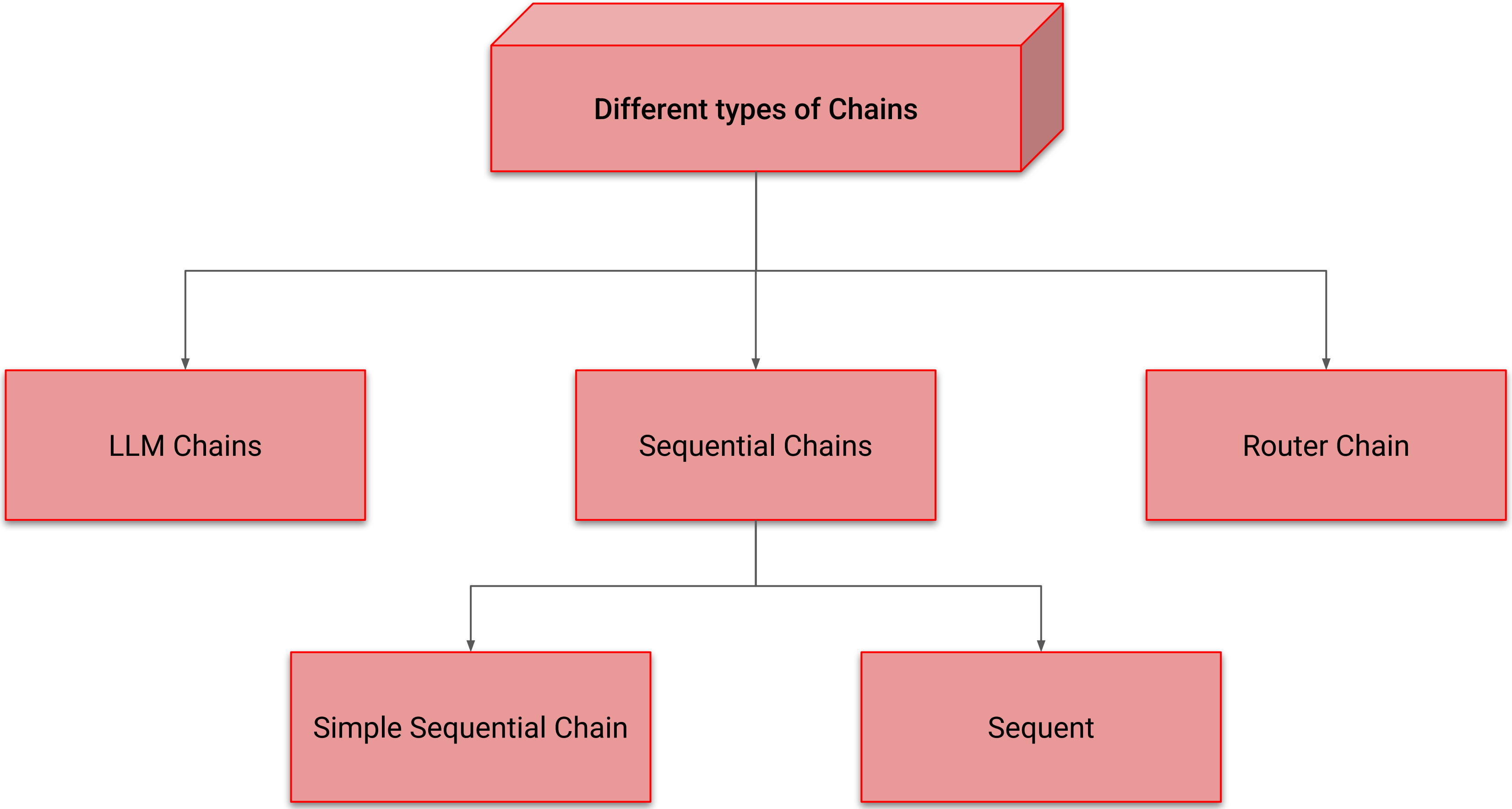
This memory allows for storing messages and then extracts the messages in a variable.

ConversationBufferWindowMemory keeps a list of the interactions of the conversation over time. It only uses the last K interactions. This can be useful for keeping a sliding window of the most recent interactions, so the buffer does not get too large.

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

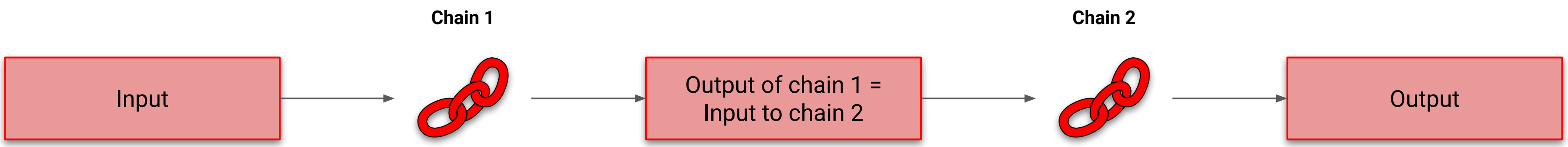
ConversationSummaryBufferMemory combines the two ideas. It keeps a buffer of recent interactions in memory, but rather than just completely flushing old interactions it compiles them into a summary and uses both. It uses token length rather than number of interactions to determine when to flush interactions.

Chains

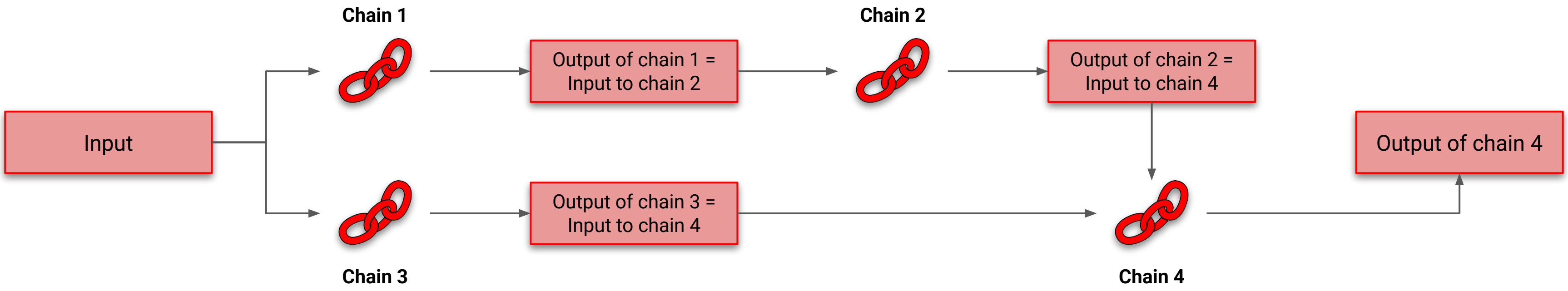

Different types of Chains
LLM Chains
Sequential Chains
Router Chain
Simple Sequential Chain
Sequent

Sequential Chains

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



Simple Sequential Chain

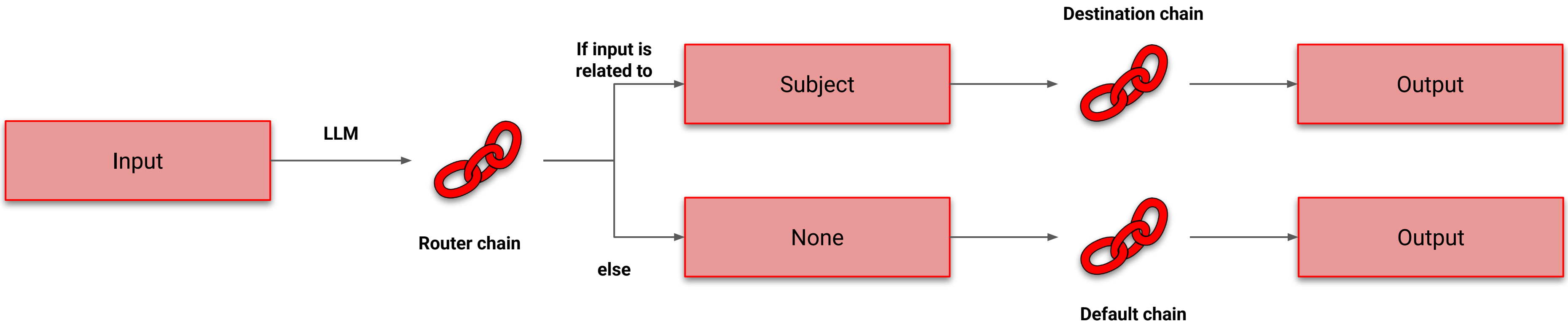


Sequential Chain

Router Chain

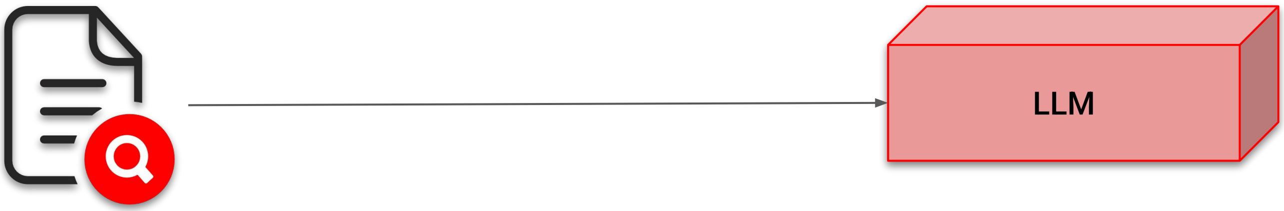
Router Chains can take in an input and redirect it to the most appropriate LLM Chain sequence.

The Router accepts multiple potential destination LLM Chains and then via a specialized prompt, the Router will read the initial input, then output a specific dictionary that matches up to one of the potential destination chains to continue processing.



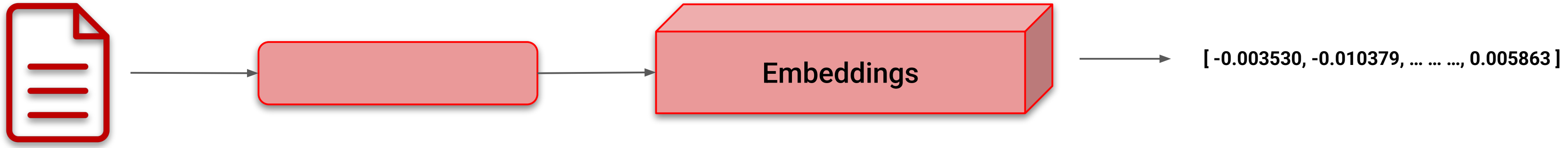
Router Chain

LLMs on Documents



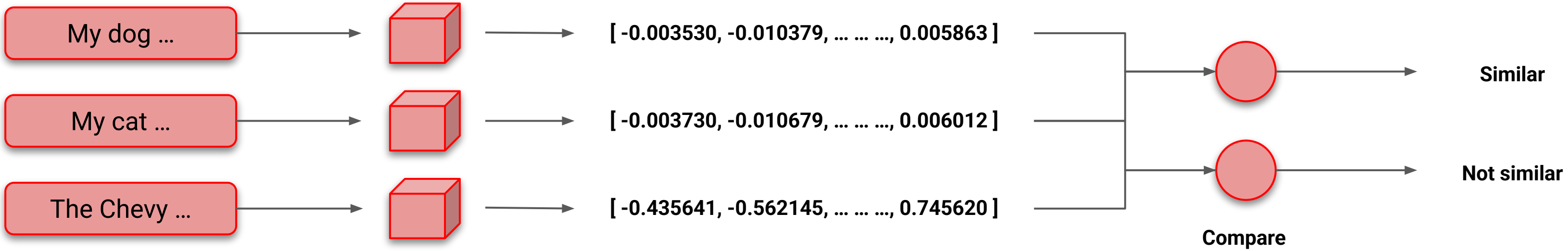
LLMs can only inspect a few thousands words at a time

Embeddings

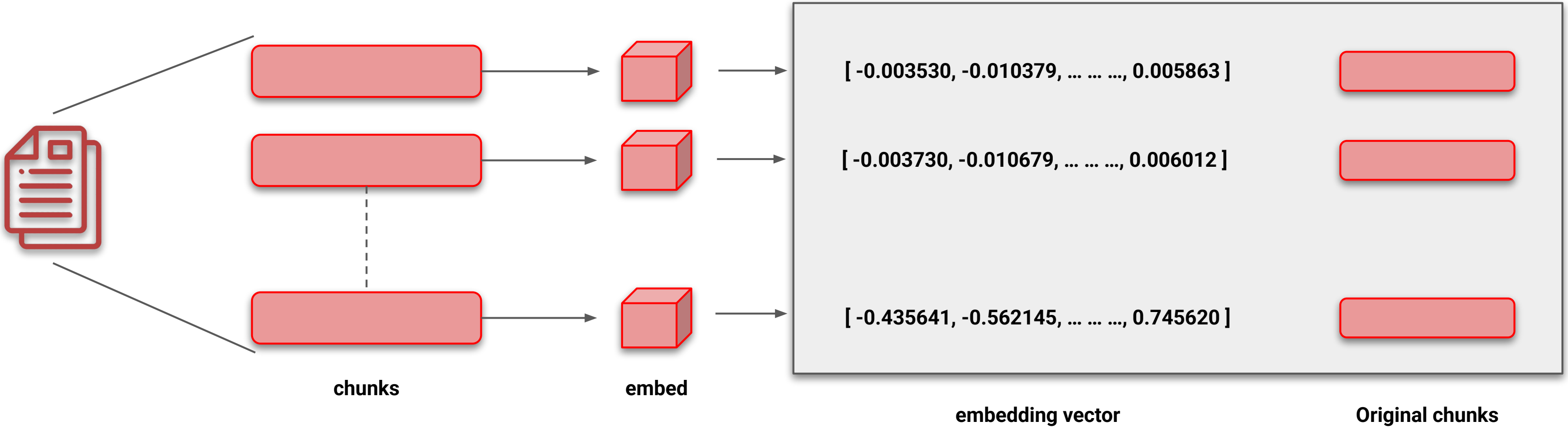


- **Embedding vector captures content / meaning**
- **Text with similar content will have similar vectors**

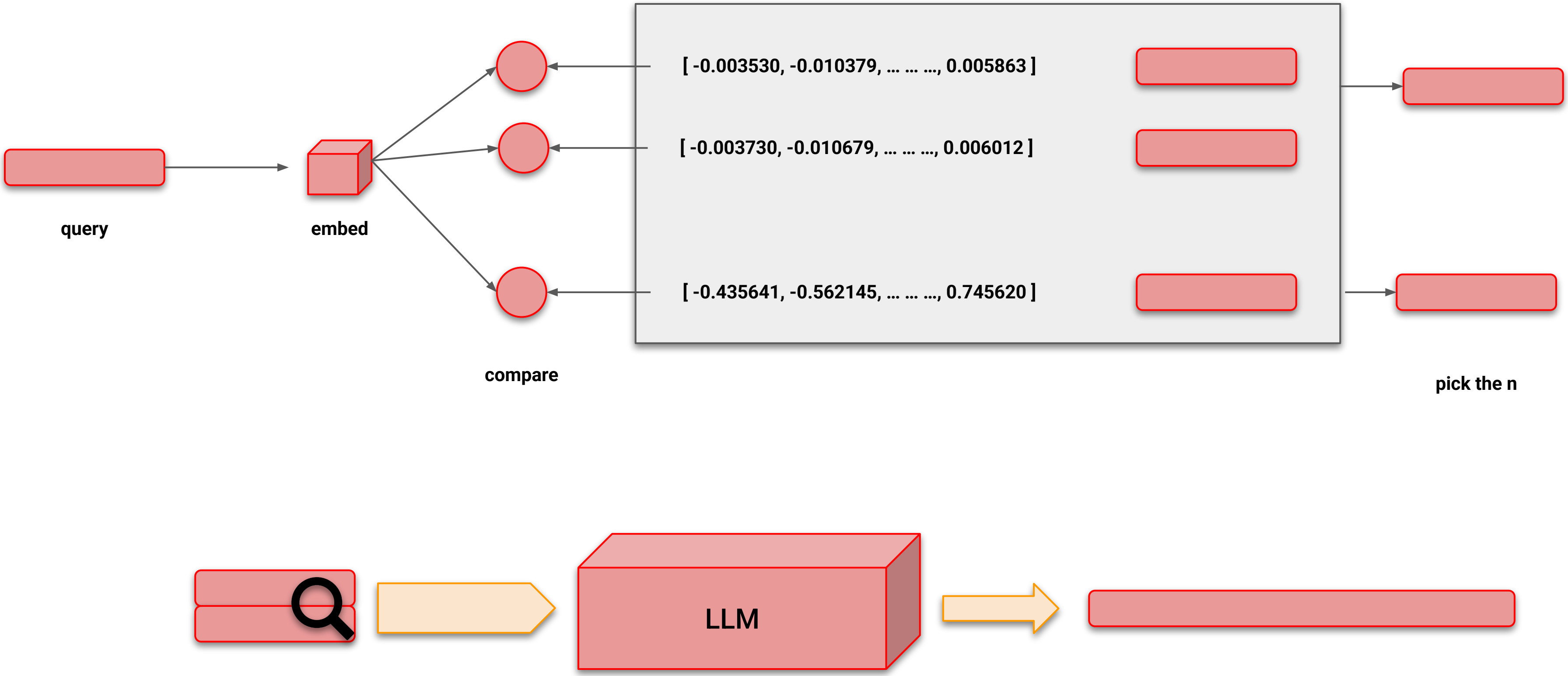
1) My dog likes to chase squirrels
2) My cat refuses to eat from a can
3) The Chevy bolt accelerates to 60 mph in 6.7 seconds



Vector Database (Create)

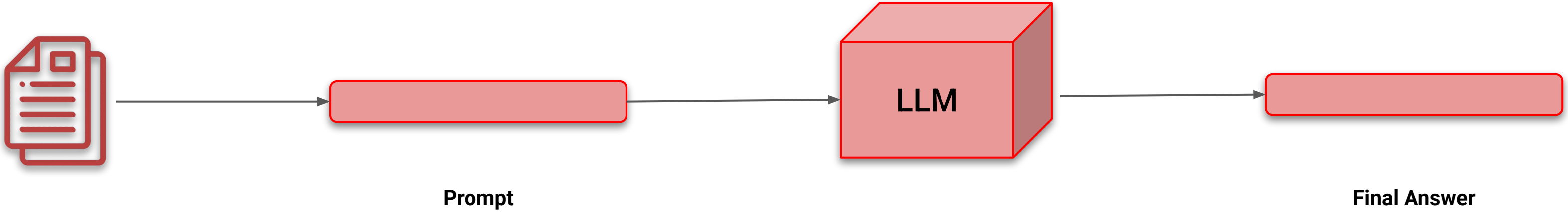


Vector Database (Index)



The returned values can now fit into LLM

Stuff Method

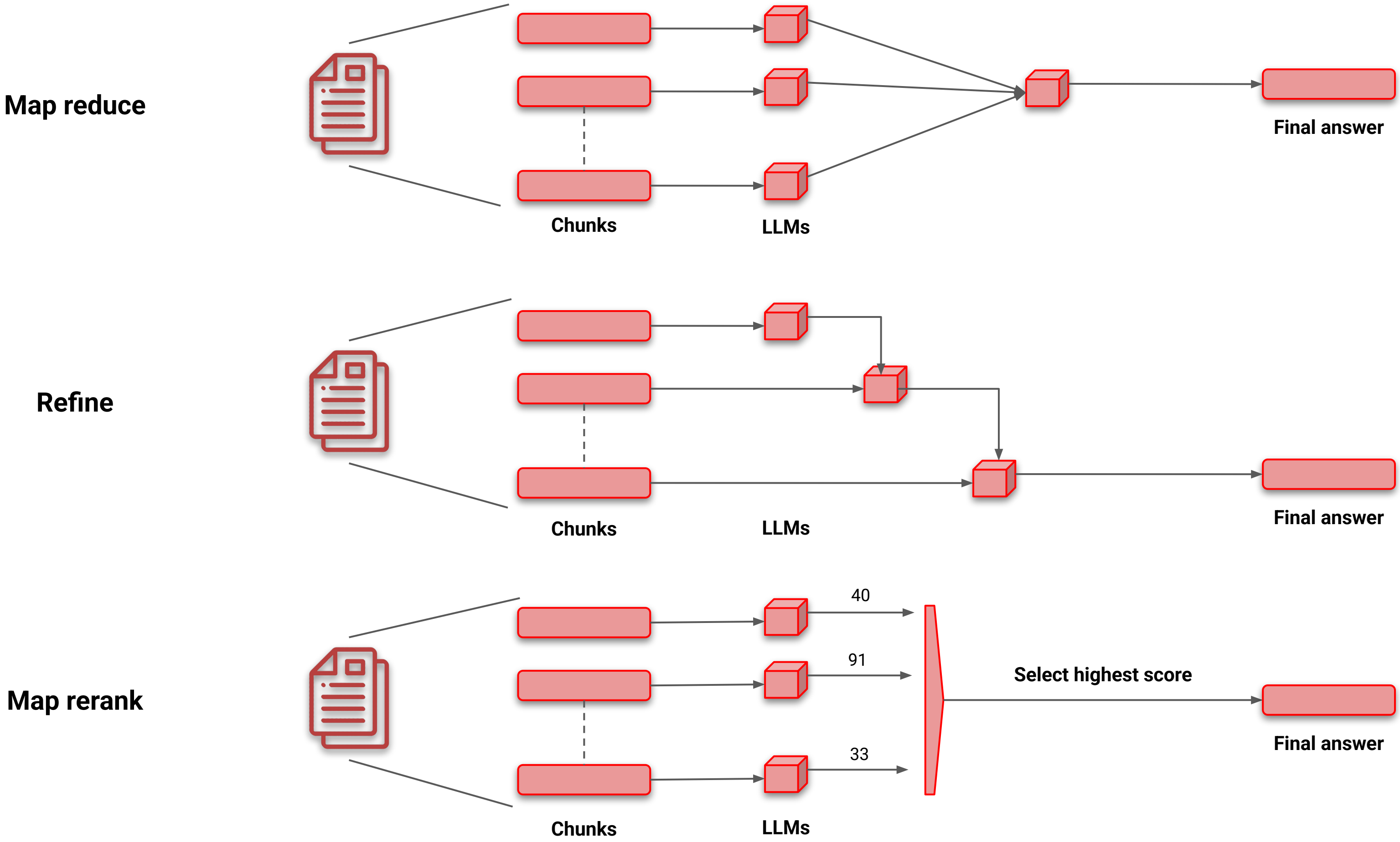


Stuffing is the simplest method. You simply stuff all the data into the prompts as context to pass to the language model

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

Additional Methods



THANK YOU