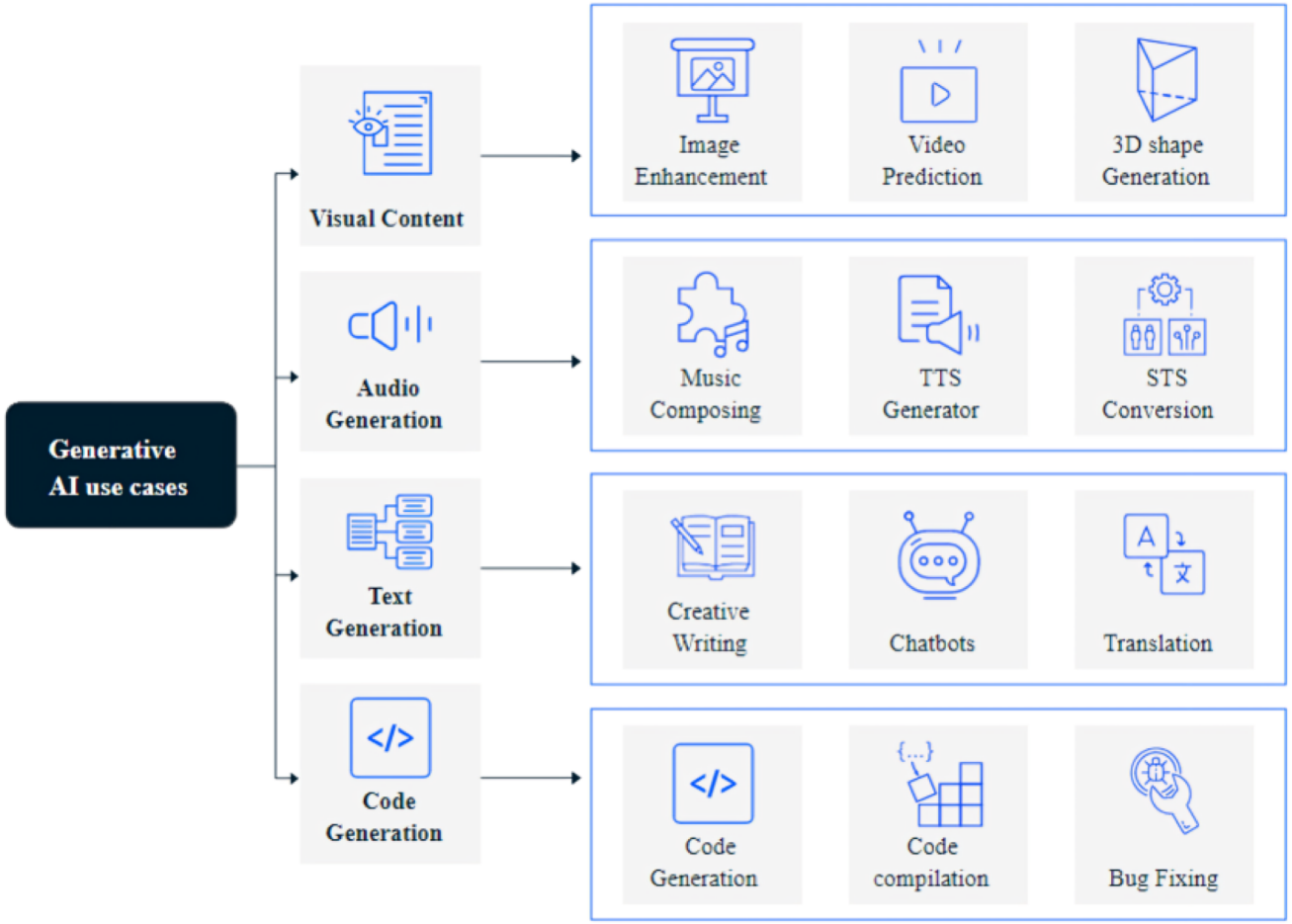
Gen AI - Azure

Day 2 - 3

Langchain

Build GAI Application

Generative AI Applications



LangChain

- LangChain is a framework for developing applications powered by language models.
- It enables applications that:
  - **Are context-aware:** connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
  - **Reason:** rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

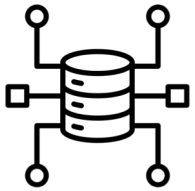
Why LangChain?

- **Components**
  - LangChain makes it easy to swap out abstractions and components necessary to work with language models.
- **Customized Chains**
  - LangChain provides out of the box support for using and customizing 'chains' - a series of actions strung together.
- **Speed**
  - This team ships insanely fast. You'll be up to date with the latest LLM features.
- **Community**
  - Wonderful discord and community support, meet ups, hackathons, etc.

LangChain Components

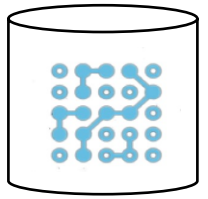
LangChain

Components

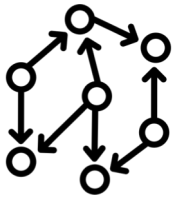


Schema

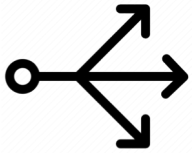
Indexes



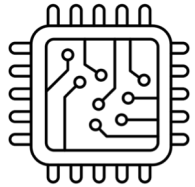
Vector Stores



Models



Text Splitters



Memory

Agents

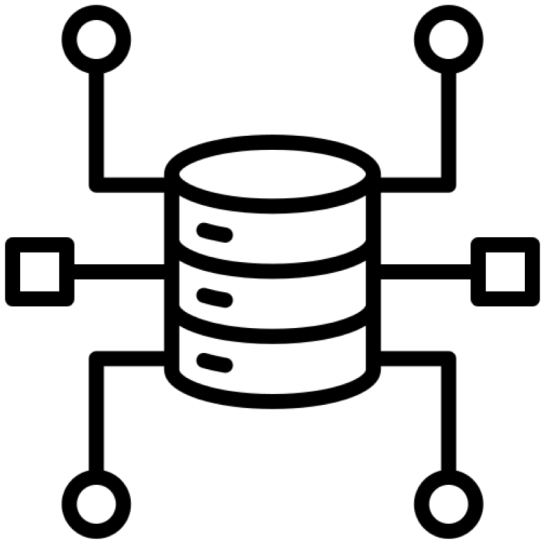
Prompts

Retrievers

Chains

Schema

- A “**Schema**" refers to structured definitions for different types of data and messages that are used to interact with Large Language Models (LLMs).
- These schemas help in organizing the data, providing consistency, and ensuring that the interactions with LLMs are well-defined and structured.



Why Do We Need Schema?

- Consistency
- Validation
- Interoperability
- Scalability
- Ease of Use

Key Components of Schema

- **Text:** A collection of characters, words, sentences, or paragraphs that form a meaningful segment of natural language. It can include letters, numbers, punctuation marks, and other symbols.

Key Components of Schema

- **Messages**: These are the fundamental building blocks for interacting with LLMs in a conversational manner. Messages are categorized into different types:
  - **SystemMessage**: Provides background context and instructions for the AI.
  - **HumanMessage**: Represents the user inputs or queries.
  - **AIMessage**: Represents the responses generated by the AI.

Key Components of Schema

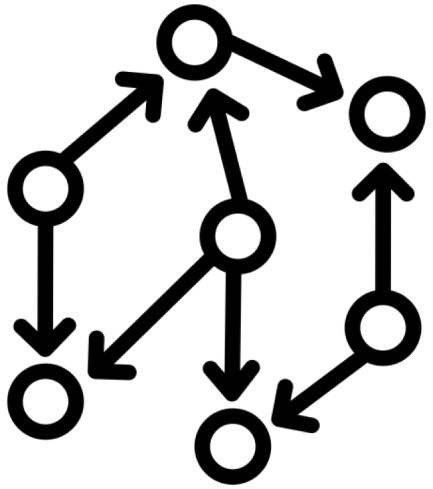
- **Documents**: These are objects that hold pieces of text along with associated metadata. Documents can be used to manage and process text data effectively.
  - **page_content**: The actual text content of the document.
  - **metadata**: Additional information about the document, such as its source, creation time, and any relevant identifiers.

Demo

Example for Conversational Messages

Models

- Models, in the context of machine learning and artificial intelligence, are mathematical representations or algorithms that have been trained on data to perform specific tasks. These tasks can include predicting outcomes, classifying data, generating text, recognizing patterns, and more.



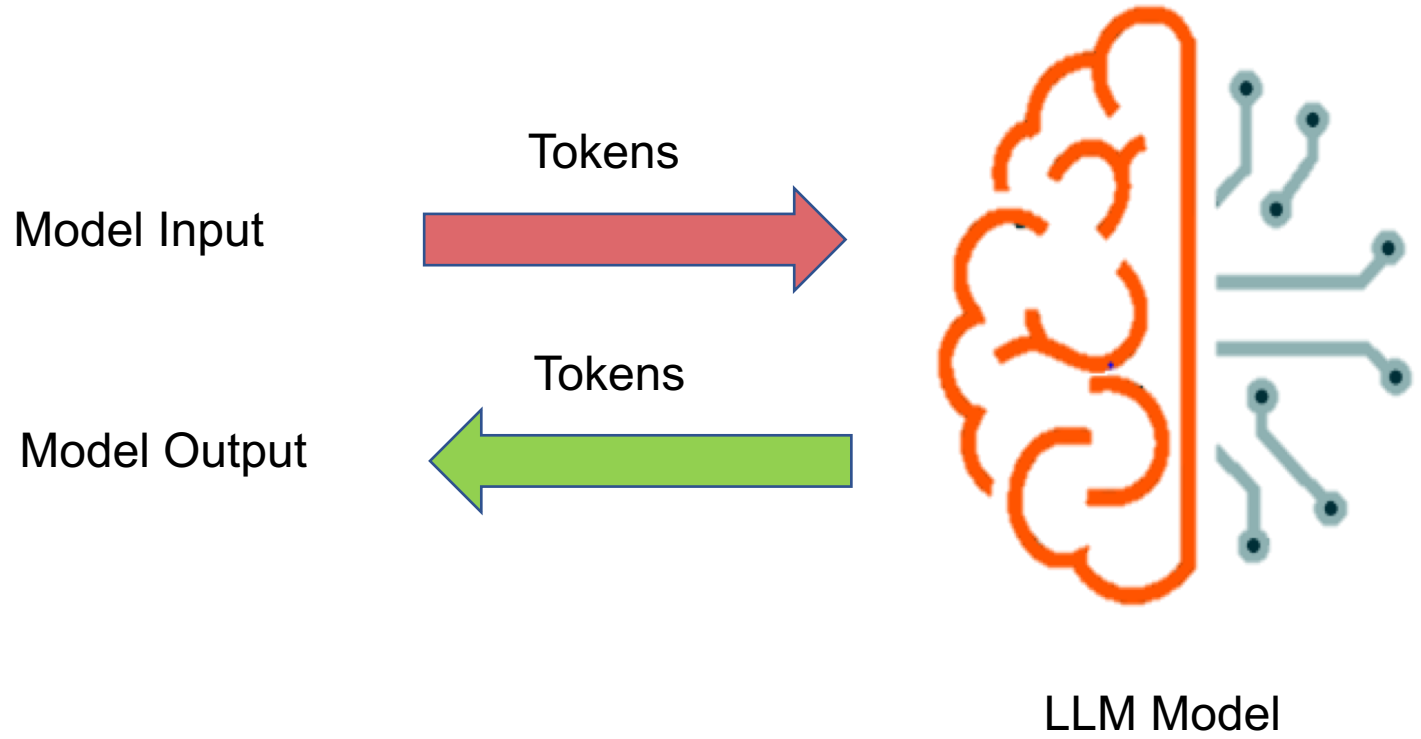
Why Do We Need Models?

- Automation and Efficiency
- Prediction and Decision Making
- Personalization
- Scalability
- Consistency and Reliability
- Advanced Capabilities

Types of Models

- **Language Models:** These models are designed to generate or comprehend text. They can take text input and produce text output, simulating human-like text generation. Examples include OpenAI's GPT (Generative Pre-trained Transformer) models.
- **Chat Models:** A subset of language models specifically tuned for conversational contexts. They handle a series of messages and can maintain context over a conversation, making them suitable for chatbots and virtual assistants.
- **Function Calling Models:** These models are fine-tuned to provide structured data outputs, which can be particularly useful for tasks like making API calls or extracting specific information in a structured format.
- **Text Embedding Models:** These models convert text into numerical vectors that represent the semantic meaning of the text. They are used in tasks such as text similarity, clustering, and search.

Standard Model Interaction



Demo

Interact with Azure OpenAI Model

Prompt Engineering

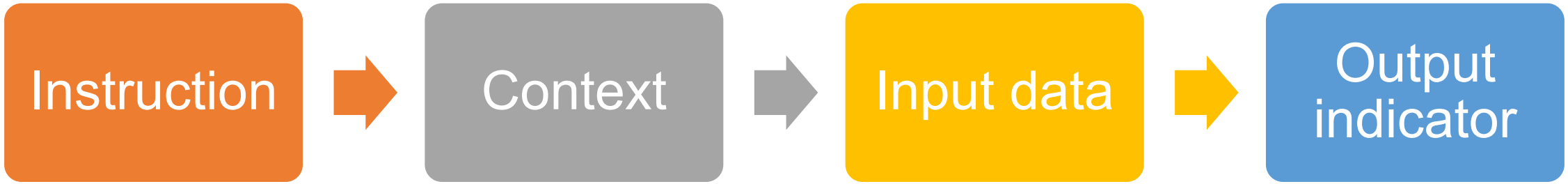
Prompt Engineering ?

- Prompt engineering is a relatively new discipline for developing and optimizing prompts to efficiently use language models (LMs) for a wide variety of applications and research topics.
- Researchers use prompt engineering to improve the capacity of LLMs on a wide range of common and complex tasks such as question answering and arithmetic reasoning.
- Developers use prompt engineering to design robust and effective prompting techniques that interface with LLMs and other tools.
- It encompasses a wide range of skills and techniques that are useful for interacting and developing with LLMs. It's an important skill to interface, build with, and understand capabilities of LLMs.

Prompt Engineering - Overview

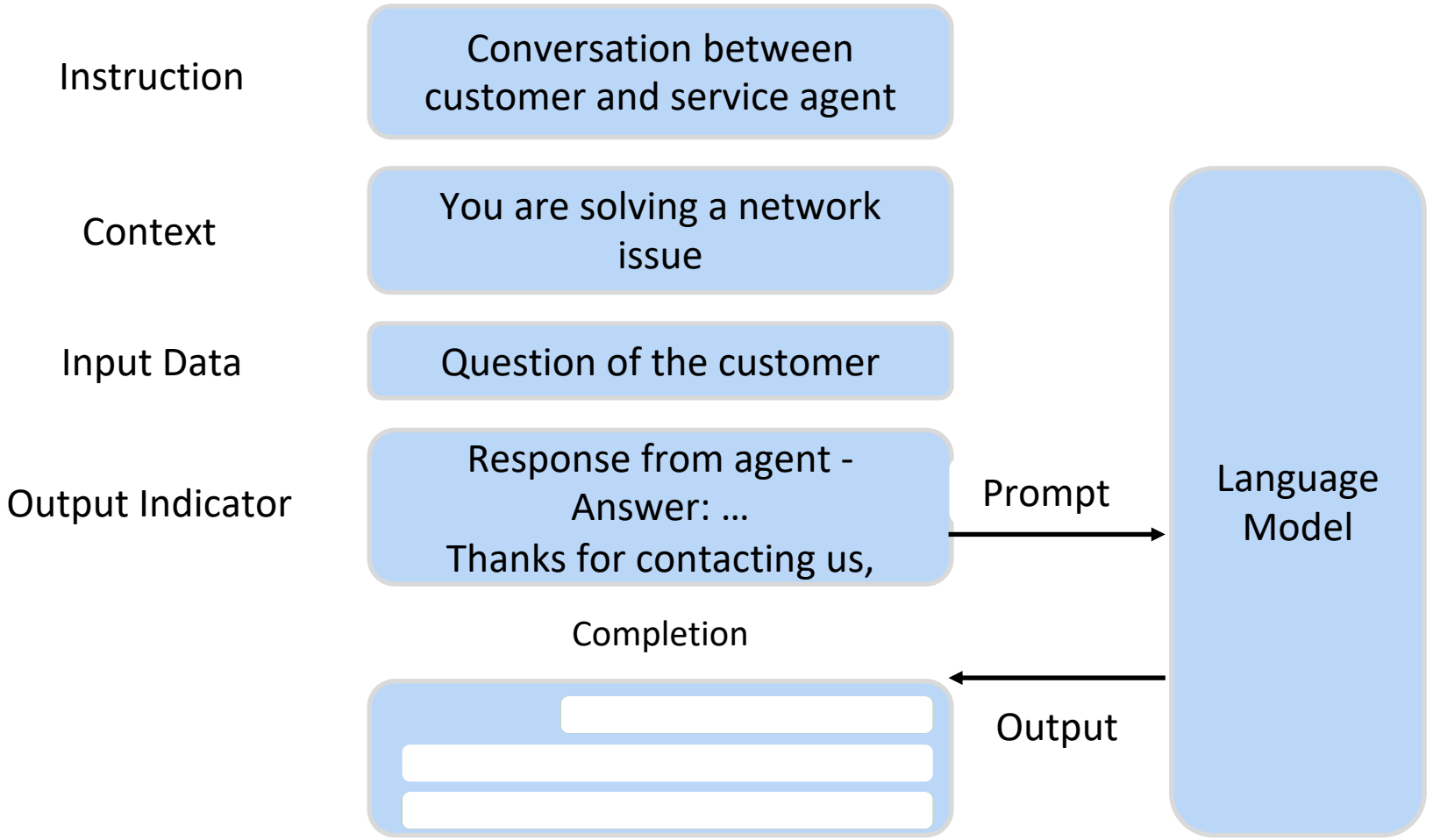
- **Definition:** Crafting & refining prompts for models like GPT-4o to produce Text, Image, Video, Audio.
- **Objective:** Enhance output quality & relevancy.
- **Application:** Chatbots, Translation, Content generation, etc.
- **Process:**
  - Choose model architecture & parameters
  - Design prompt structure & format
  - Select task & training data
  - Fine-tune model with prompt & data

Elements of a Prompt



Basic of Prompting

Prompt:



Prompt Settings

For Respective Models

Prompt Settings Samples

Setting	Description
Temperature	Controls response creativity; higher = more random, lower = more focused.
Max Tokens	Limits the total number of tokens (words/pieces) in the output.
Top P	Samples from top probability tokens until their sum reaches P; lower = more focused.
Max Response	Caps the maximum length for single response.
Timeout	Sets the maximum time (in ms) to wait before cancelling the response.
Token Per Second	Controls how quickly tokens are generated per second.
Frequency Penalty	Penalizes repetition of the same words to encourage diversity.
Presence Penalty	Discourages reusing previously mentioned words or topics.

Temperature

- **Definition:**

  Temperature is a parameter that controls the randomness of the model's responses. Higher values (e.g., 0.8) make the output more creative and diverse, while lower values (e.g., 0.2) make the responses more focused and deterministic.

- **Suggested Range:**

  Adjust based on preference, with higher values for creative responses and lower values for more focused and controlled answers.

Max Tokens

- **Definition:**

  Max tokens limit the length of the response generated by the model. Setting a maximum number of tokens can help in controlling the length of the output.

- **Suggested Range:**

  Set based on desired response length; however, be cautious not to set it too low, as it may result in incomplete or nonsensical responses.

Timeout

- **Definition:**

  Timeout determines the maximum duration allowed for the model to generate a response. It helps manage the time taken for API calls.

- **Suggested Range:**

  Adjust based on your application's requirements and acceptable response times.

Tokens Per Second (TPS)

- **Definition:**

  Tokens per second is a measure of the speed at which the model processes input and generates output. It's important to stay within the model's maximum token limit.

- **Suggested Range:**

  Stay within the token limit specified for the model to avoid incomplete responses.

Examples of Prompts

Examples of Prompts

- Text Summarization
- Information Extraction
- Question Answering
- Text Classification
- Conversation
- Code Generation
- Reasoning

Prompt Engineering Techniques

Prompt Engineering Techniques

- Zero shot
- Few shot
- Chain of Thought (CoT)
- Self consistency
- Generated Knowledge Prompting
- Prompt Chaining
- Tree of Thoughts(ToT)
- Automatic Reasoning and Tool use (ART)
- Directional Stimulus
- ReAct Prompting

Zero shot

Prompt :

Classify the text into neutral, negative or positive.
Text: I think the vacation is okay.
Sentiment:

Output: Neutral

Few shot

Prompt :

This is awesome! // Negative
This is bad! // Positive
Wow that movie was rad! // Positive
What a horrible show! //

Output: Negative

Chain of Thought (CoT)

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9.
The answer is 9. ✓

ReAct Prompting

- The React Prompting technique involves having a back-and-forth interaction with the AI model, where you provide feedback or react to its responses.

- This technique allows you to guide the AI's output through iterative refinement based on your reactions.

ReAct Prompting

Prompt : Generate a Python function that calculates the sum of all even numbers in a list. Please make sure the function is efficient and handles empty lists.

Output:

```
def sum_even_numbers(numbers):
    total = 0
    for num in numbers:
        if num % 2 == 0:
            total += num
    return total
```

Prompt Template

Prompt Template

- A prompt template is a structured framework that allows for dynamic generation of prompts based on predefined patterns and placeholders. It typically includes fixed text and variables that can be filled with specific values at runtime. Prompt templates are useful when generating multiple prompts with similar structures but varying content or style.

Key Components of a Prompt Template

- **Template String**: The base text with placeholders for variables.
- **Variables**: Parameters that can be substituted into the template to create specific instances of the prompt.
- **Formatting Logic**: Rules or methods for inserting variables into the template.

Example

```
template_string = """Translate the text that is delimited by triple backticks
into a style that is {style}. text: ```{text}``` """

from langchain_core.prompts import ChatPromptTemplate
prompt_template = ChatPromptTemplate.from_template(template_string)

employee_style = """British English in a calm and respectful tone"""

employee_email = """
Objet : Demande de Congé
Cher [Nom du Responsable], . . .
Cordialement,
John """

employee_messages = prompt_template.format_messages(style=employee_style,
text=employee_email)
```

Example

```
Print(employee_messages)


[HumanMessage(content="Translate the text that is delimited by triple
backticks into a style that is British English in a calm and respectful tone\n. text:
```\nObjet : Demande de Congé\n\nCher [Nom du Responsable],\n\nJe m'appelle John et je travaille au sein de la
société XYZ.\nJ'aimerais solliciter une demande de congé pour partir en vacances.\nSerait-il possible de discuter
des dates qui conviendraient le mieux pour l'équipe et l'entreprise?\n\nJe vous remercie par avance pour votre
compréhension et j'attends votre retour.\n\nCordialement,\nJohn\n\n```\n", additional_kwargs={},
response_metadata={})]
```

Above message will be the Input for LLM Model.
```

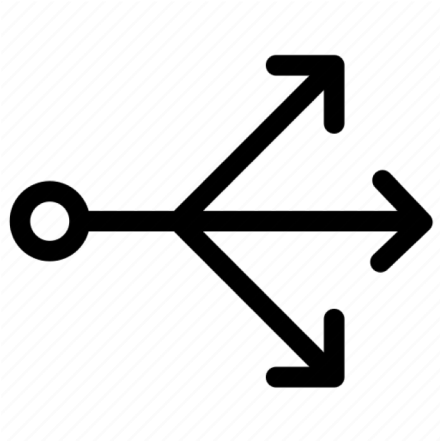
Demo

Case Study: Email Response Service For Customer Support

Text Splitters

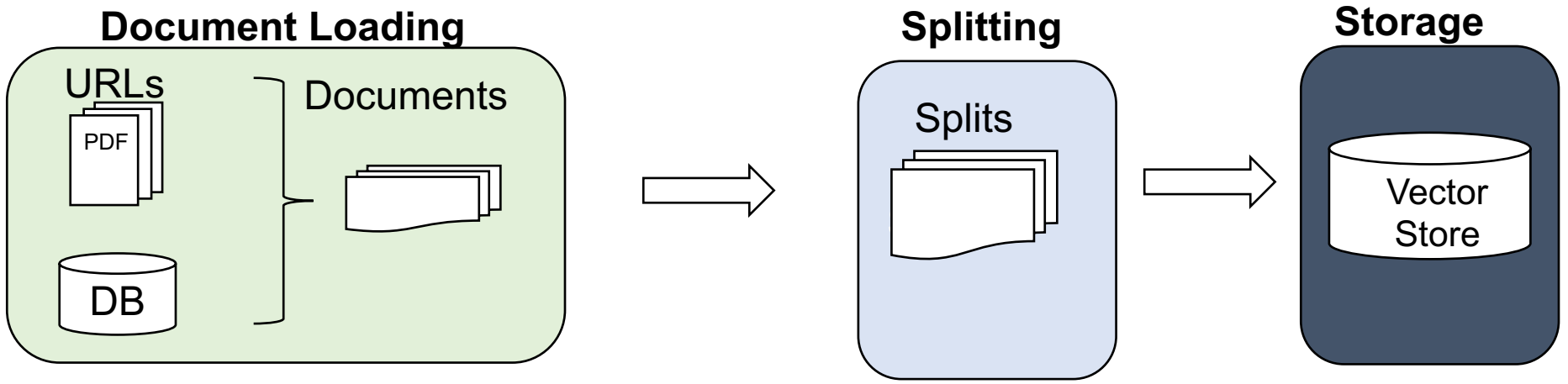
Text Splitting

- Text splitting is crucial because it ensures that semantically relevant content is grouped together within the same chunk. This is particularly important when answering questions or performing other tasks that rely on the contextual information present in the documents.



Document Splitting

- Splitting Documents into smaller chunks
- Retaining meaningful relation



Document Splitting

....
on this model. The Toyota Camry has a head-snapping 80 HP and an eight-speed automatic transmission that will
....
Chunk 1: on this model. The Toyota Camry has a head-snapping
Chunk 2: 80 HP and an eight-speed automatic transmission that will
Question: What are the specifications of the Camry?

Example Splitter

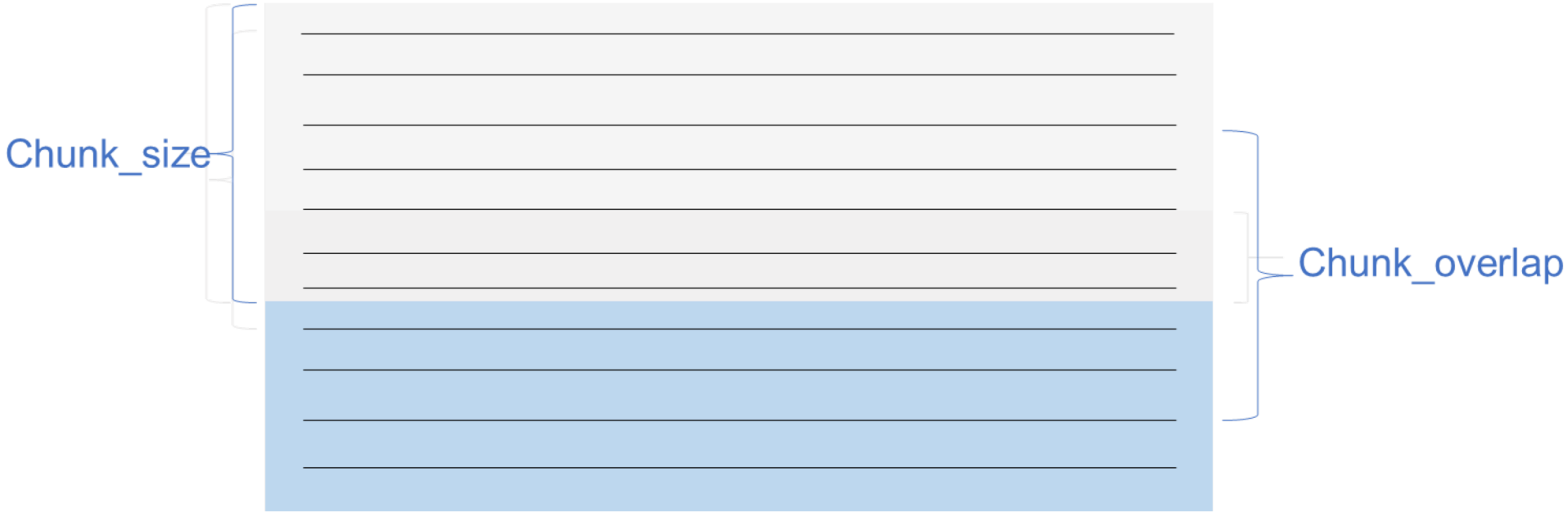
langchain.text_splitter.CharacterTextSplitter

(
separator: str = "\n\nz”
chunk_size=4000,
chunk _overlap=200,
length_function=<builtin function len>,
)

Methods:

- **create_documents()** - Create documents from a list of texts.
- **split_documents()** - Split documents.

Example Splitter



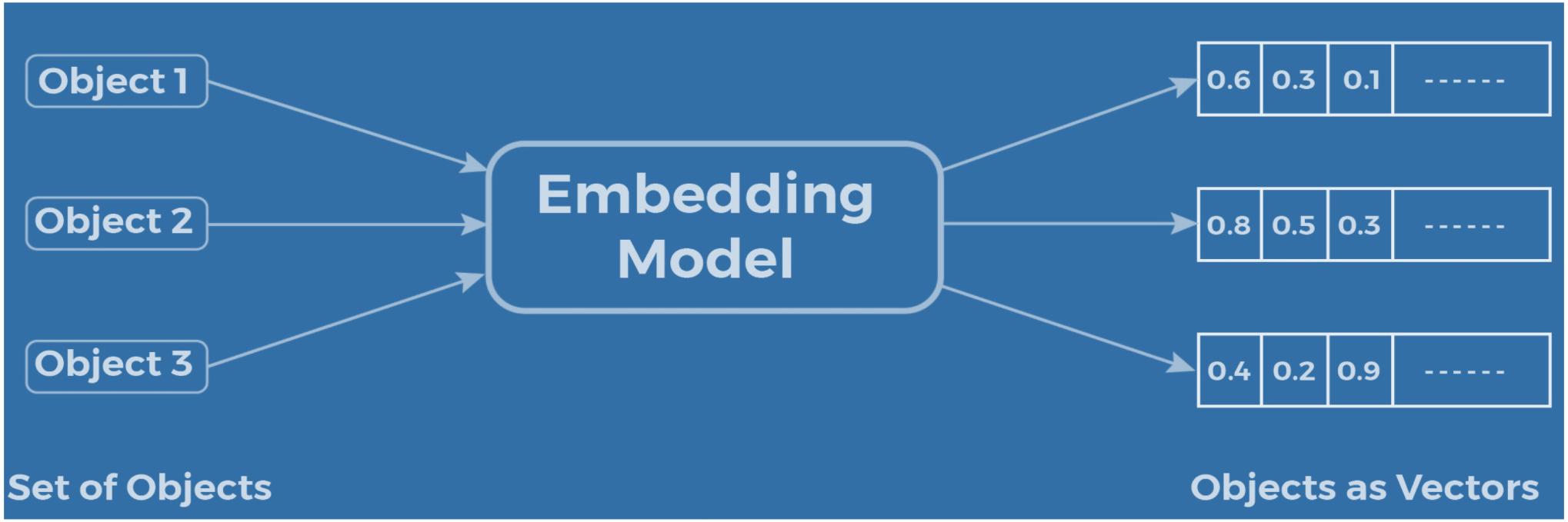
Types of Text Splitters

langchain.text_splitter.

- **CharacterTextSplitter()** - Implementation of splitting text that looks at characters.
- **MarkdownHeaderTextSplitter()** - Implementation of splitting markdown files based on specified headers.
- **TokenTextSplitter()** - Implementation of splitting text that looks attokens.
- **SentenceTransformersTokenTextSplitter()** - Implementation ofsplitting text that looks at tokens.
- **RecursiveCharacterTextSplitter()** - Implementation of splitting textthat looks at characters. Recursively tries to split by differentcharacters to find one that works.

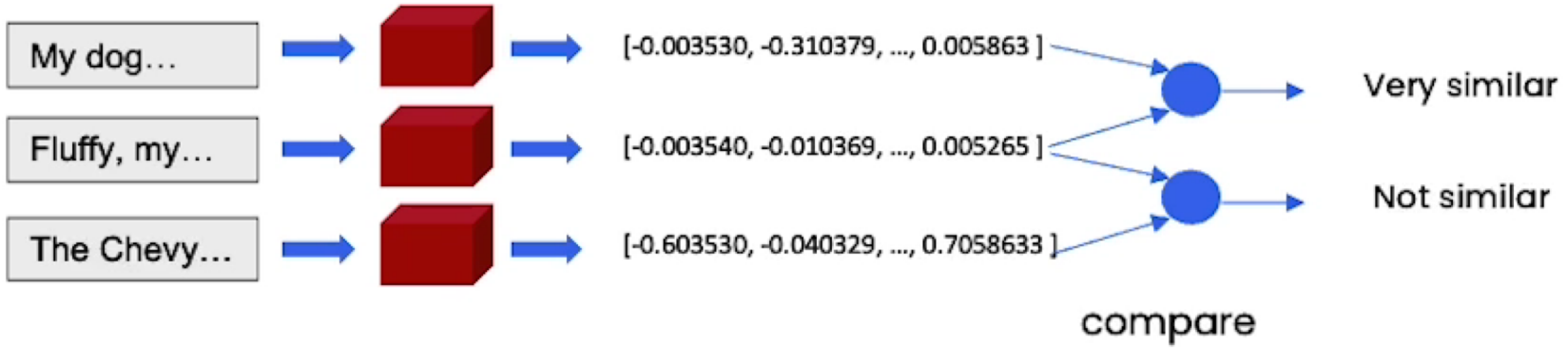
Embeddings

Embeddings



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

Embeddings

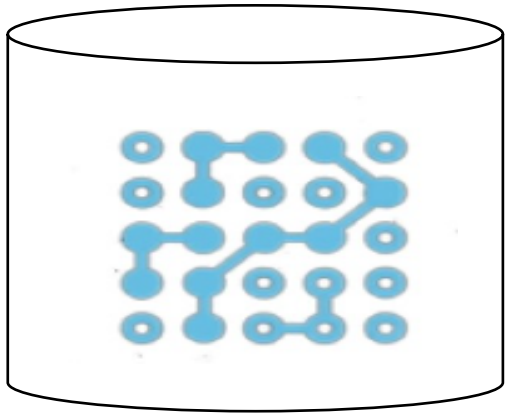


1. My dog Rover likes to chase squirrels.
2. Fluffy, my cat, refuses to eat from a can.
3. The Chevy Bolt accelerates to 60 mph in 6.7 seconds.

Vector Store

Vector Store

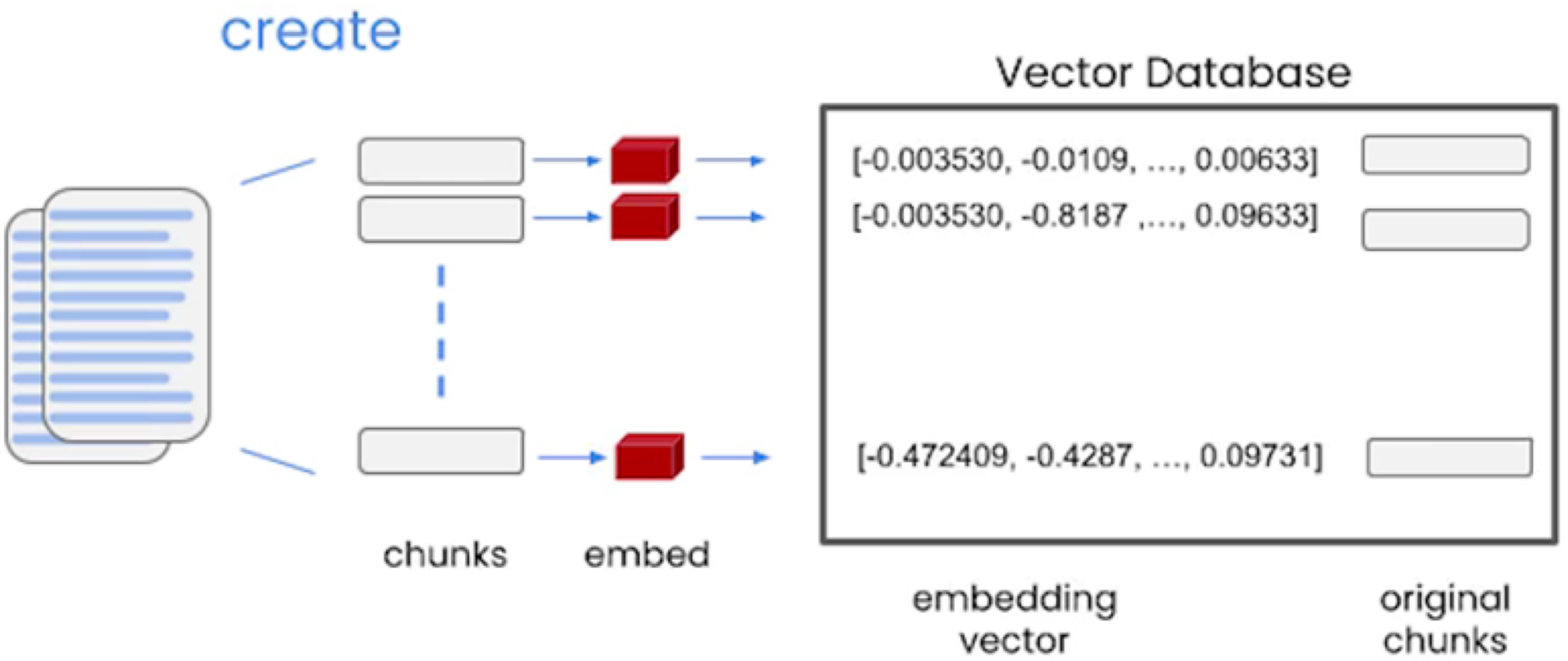
- A vector store is a database designed to store and manage numerical vectors, such as embeddings, for efficient retrieval and similarity search. It enables quick and accurate matching of vectors, facilitating tasks like nearest neighbor search, clustering, and recommendation systems based on vector similarity.



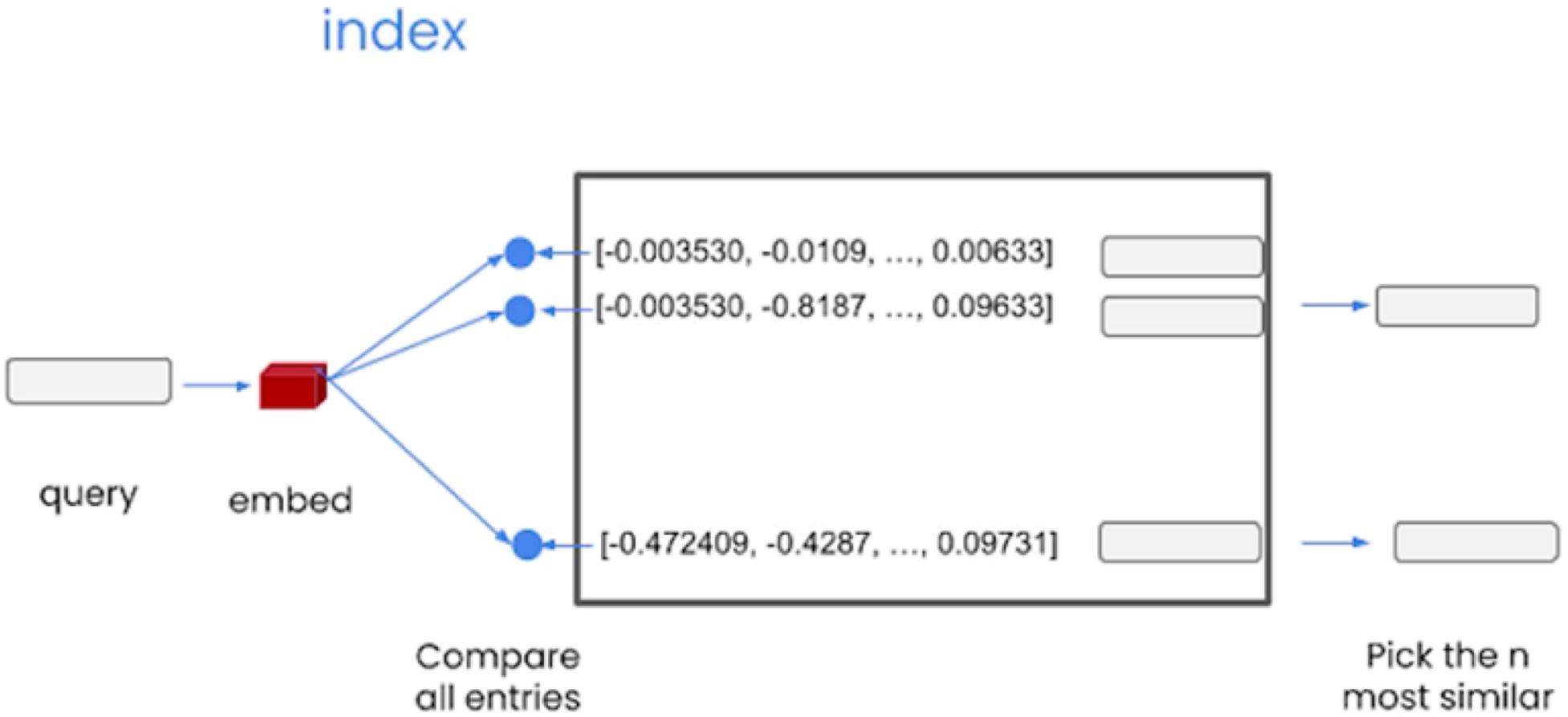
Vector Stores

- One of the most common ways to store and search over unstructured data is to embed it and store the resulting embedding vectors, and then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query.
- A vector store takes care of storing embedded data and performing vector search for you.

Vector Store



Vector Store

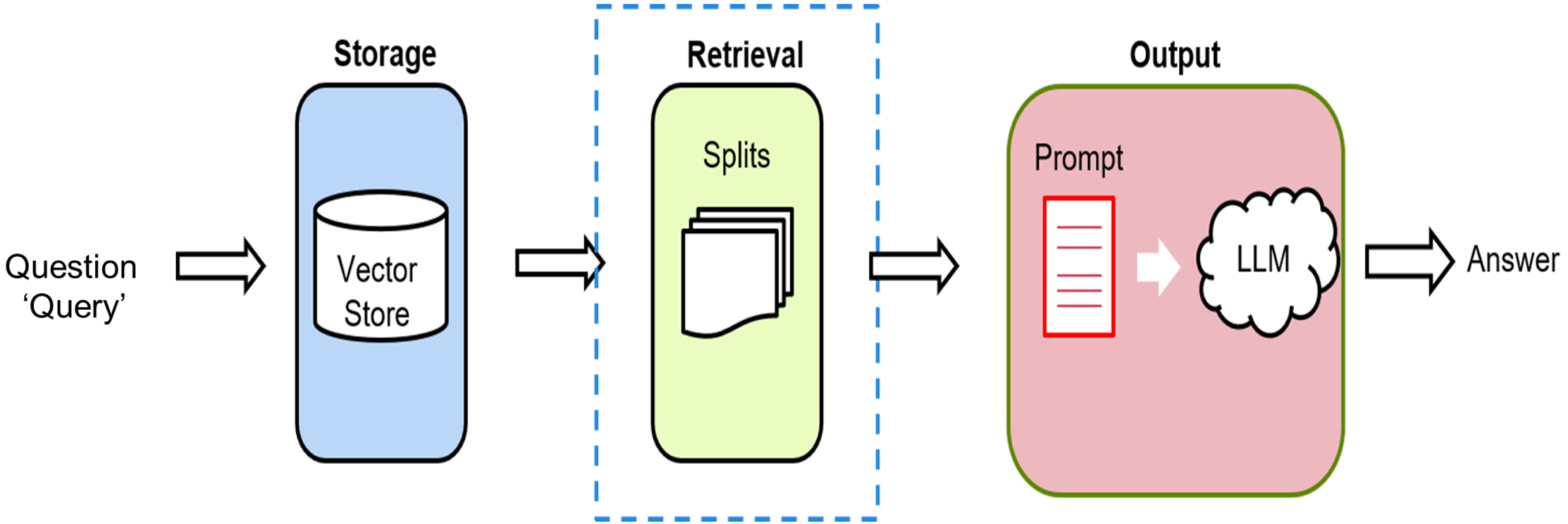


Retrieval

Retrieval

- Retrieval refers to the process of accessing and fetching relevant pieces of information from a collection of documents or data sources based on a query.
- This involves using techniques like similarity search and embedding-based methods to find and return the most pertinent documents or text segments that match the query's context and content.

Retrieval


Storage
Retrieval
Output
Question
'Query'
Vector
Store
Splits
Prompt
LLM
Answer

Retrieval

- Accessing/indexing the data in the vector store
  - Basic semantic similarity
  - Maximum marginal relevance
  - Including Metadata
- LLM Aided Retrieval

Why Do We Need Retrieval?

- Enhanced Information Access
- Improved Response Quality
- Efficiency and Performance
- Flexibility and Customization
- Support for Complex Queries
- Dynamic Knowledge Updating

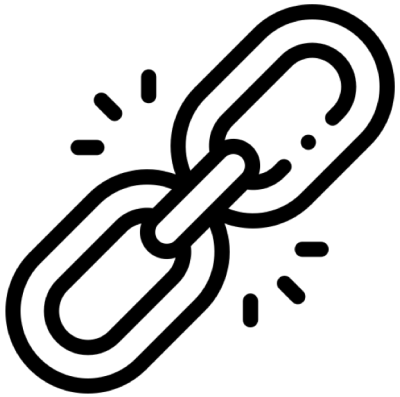
Types of Retrevial

- Vectorstore Retrieval
- Similarity Search
- Maximum marginal Relevance
- Metadata Filtering
- Metadata using Self-Query Retriever

Chains

Langchian Chains

- Chains are a fundamental concept that allows you to execute complex tasks in a structured and efficient way.
- Chains are invaluable due to their capacity to effortlessly blend diverse components, shaping a singular and coherent application. Through the creation of chains, multiple elements can seamlessly come together.



Why Do We Need Chains?

- Enhanced Complexity Handling
- Modularity and Reusability
- Improved Accuracy and Efficiency
- Flexibility and Scalability
- Contextual Awareness
- Error Handling and Recovery

LLM Chain - The simplest chain

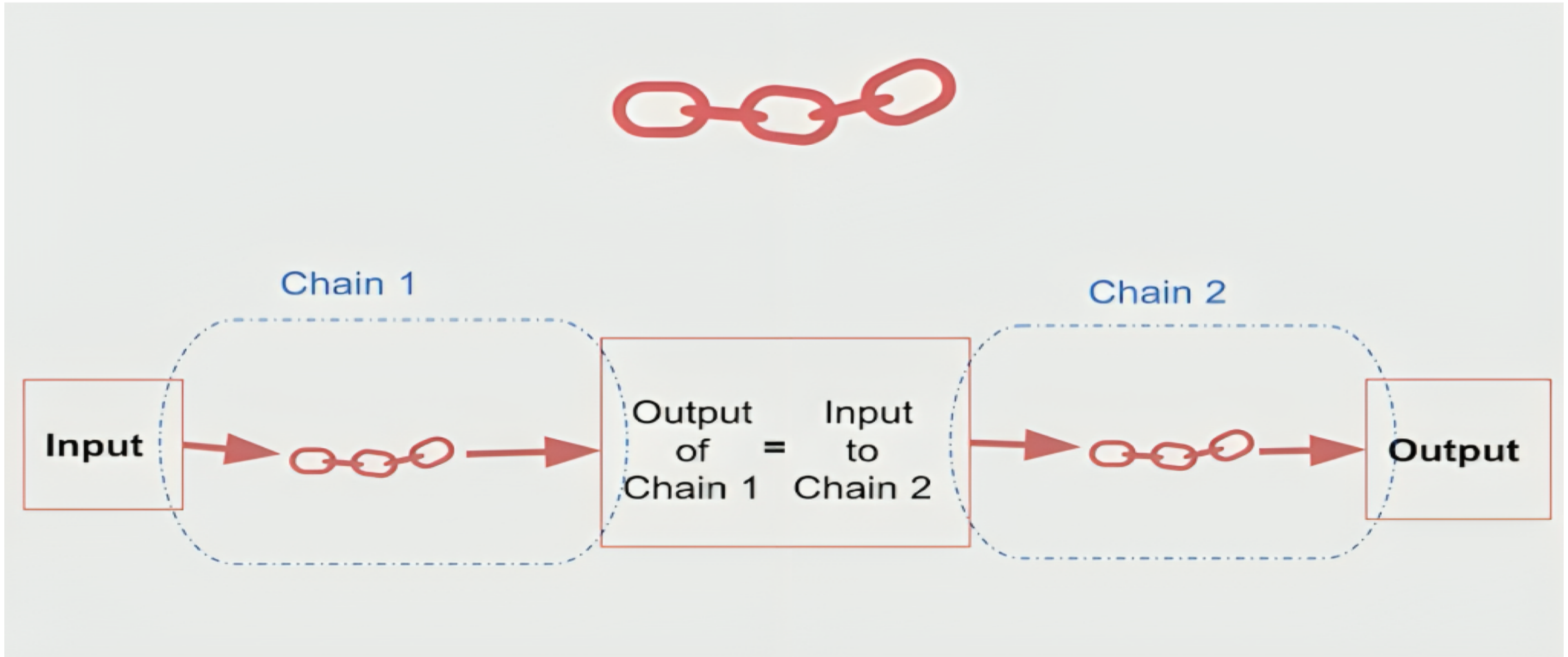
- The LLMChain is a foundational system that includes a PromptTemplate, an OpenAI model (such as a Large Language Model or a ChatModel), and optionally, an output parser. It operates by transforming input parameters using the PromptTemplate into a coherent prompt, which is then fed into the model. The resulting output is further refined and formatted into a usable form by the OutputParser, if provided. This structured approach ensures effective utilization of language models for various applications, enhancing their functionality and utility.


```
prompt = ChatPromptTemplate.from_template(
    "What is the best name to describe a company
that makes {product}?"
)
```

```
chain = LLMChain(llm=llm,
prompt=prompt)
```

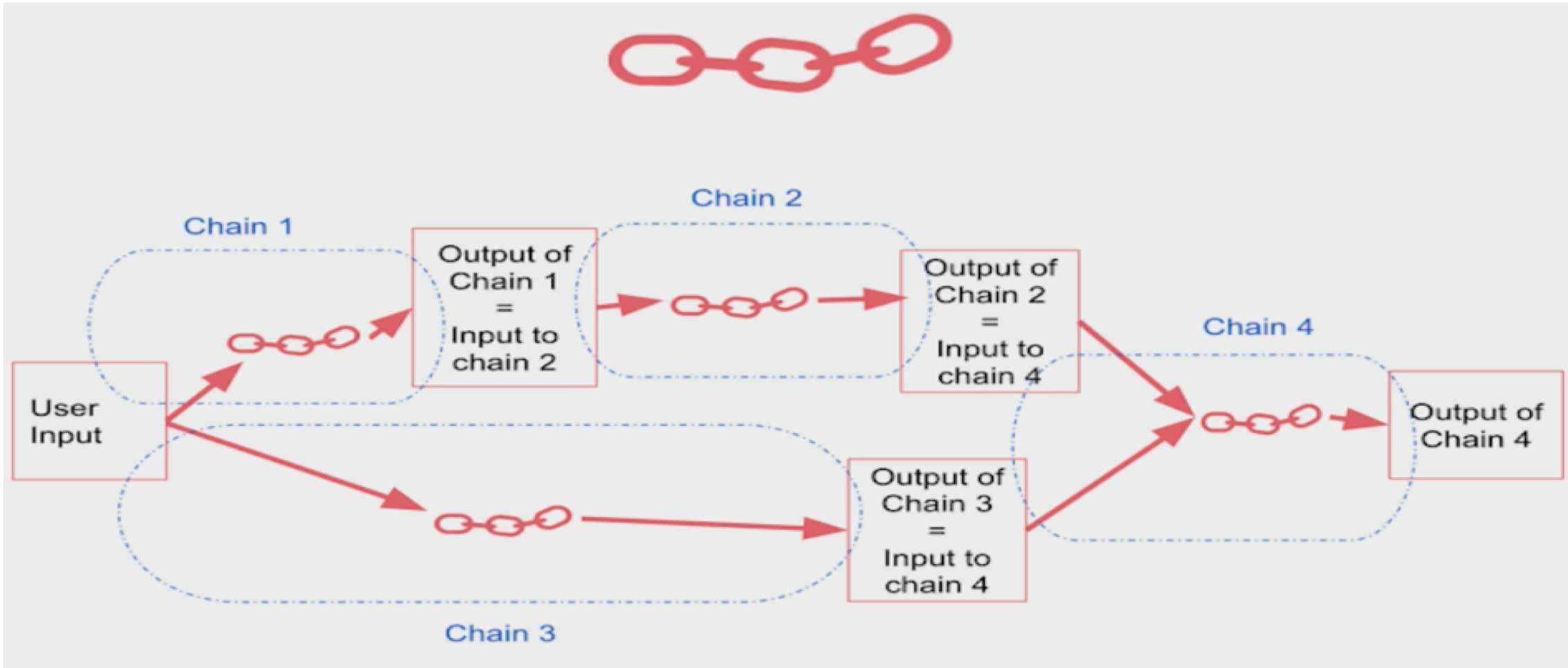
SimpleSequentialChain

- Simple Sequential Chains allow for a single input to undergo a series of coherent transformations, resulting in a refined output. This sequential approach ensures systematic and efficient handling of data, making it ideal for scenarios where a linear flow of information processing is essential


Chain 1
Chain 2
Input
Output of Chain 1 = Input to Chain 2
Output

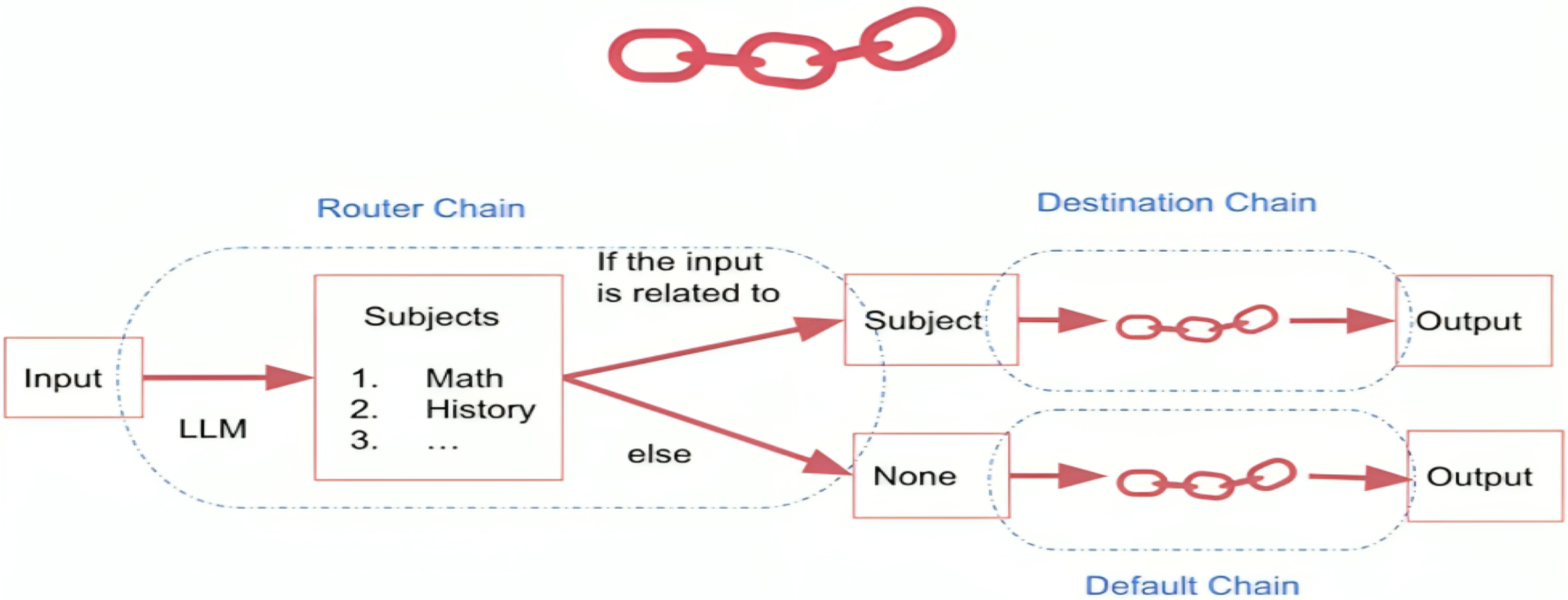
SequentialChain

- A sequential chain is a chain that combines various individual chains, where the output of one chain serves as the input for the next in a continuous sequence. It operates by running a series of chains consecutively.


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

Router Chain

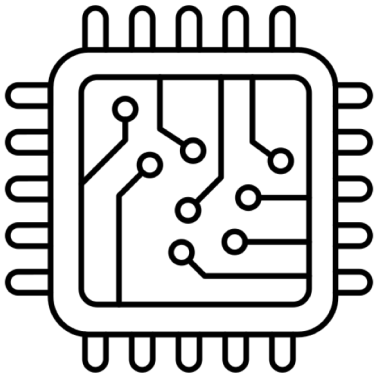
- The Router Chain is used for complicated tasks. If we have multiple subchains, each of which is specialized for a particular type of input, we could have a router chain that decides which subchain to pass the input to.


Router Chain
Destination Chain
Input
LLM
Subjects
1. Math
2. History
3. …
If the input is related to
else
Subject
None
Output
Output
Default Chain

Memory

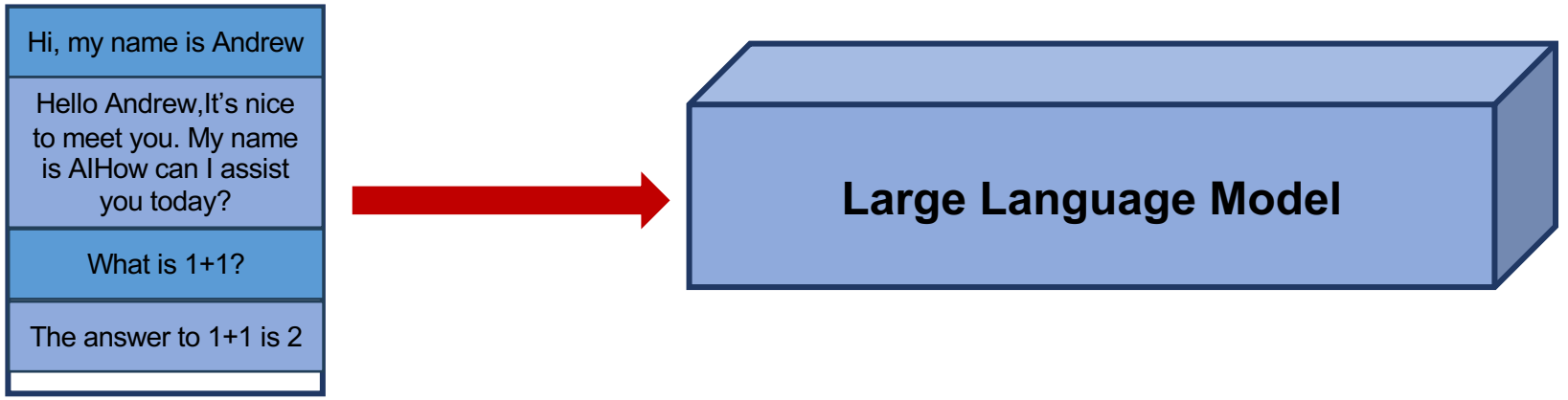
Memory

- The memory allows a Large Language Model (LLM) to remember previous interactions with the user. By default, LLMs are stateless meaning each incoming query is processed independently of other interactions.



Memory

- Large Language Models are 'stateless'
  - Each transaction is independent
- Chatbots appear to have memory by providing the full conversation as 'context'



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.

Memory Types

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.

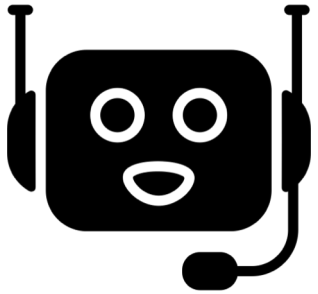
Additional Memory Types

- 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)

Agents

Agents

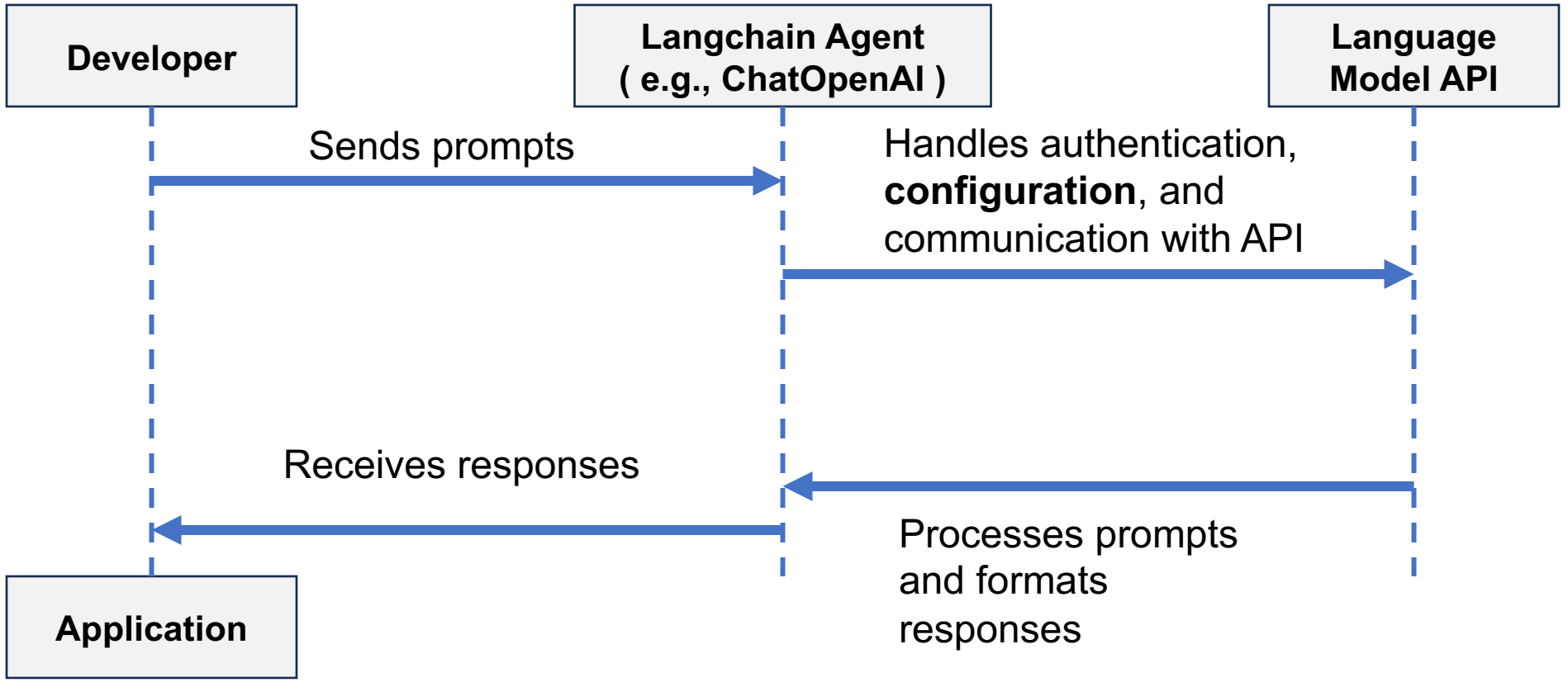
- They are specialized components within the LangChain framework that act as intelligent assistants, using LLMs to process natural language input and then orchestrating a sequence of actions to achieve a desired outcome.



Why Do We Need Agents?

- Automation of Repetitive Tasks
- Consistency in Responses
- Scalability
- Data Processing and Analysis
- Integration with Systems
- Enhanced User Experience
- Cost Efficiency

Agents


Developer
Langchain Agent
(e.g., ChatOpenAI)
Language
Model API
Sends prompts
Handles authentication,
configuration, and
communication with API
Receives responses
Processes prompts
and formats
responses
Application

RAG as an Agent

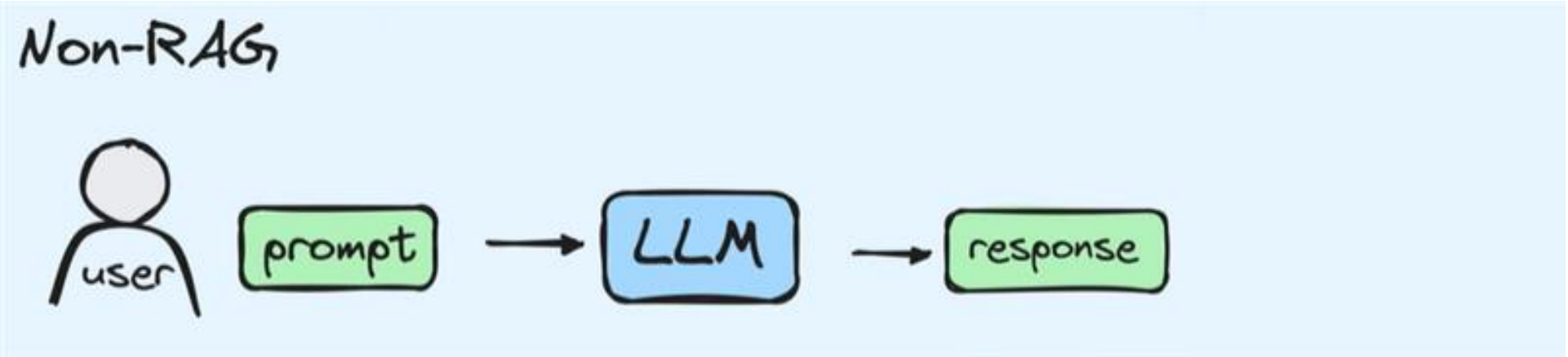
- RAG can be implemented to act as an intelligent assistant, by custom configuring with proprietary data.

Retrieval Augmented Generation (RAG)

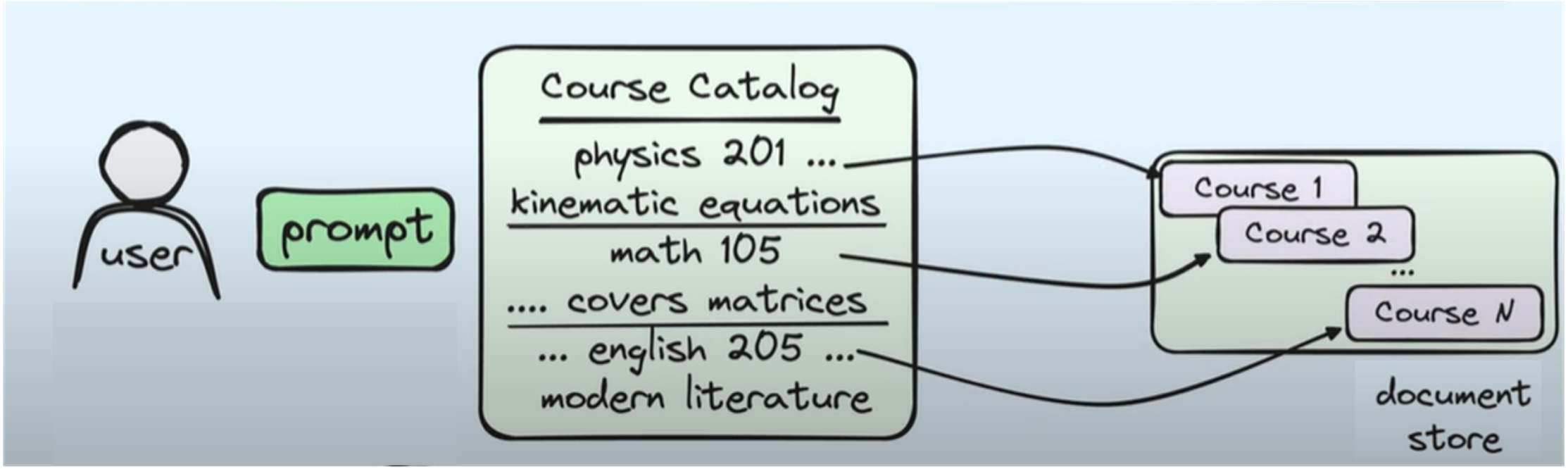
- RAG (Retrieval Augmented Generation) is a technique for augmenting LLM knowledge with additional, often private or real-time, data.

- LLMs are trained on enormous bodies of data but they aren’t trained on your data. RAG solves this problem by adding your data to the data LLMs already have access to.

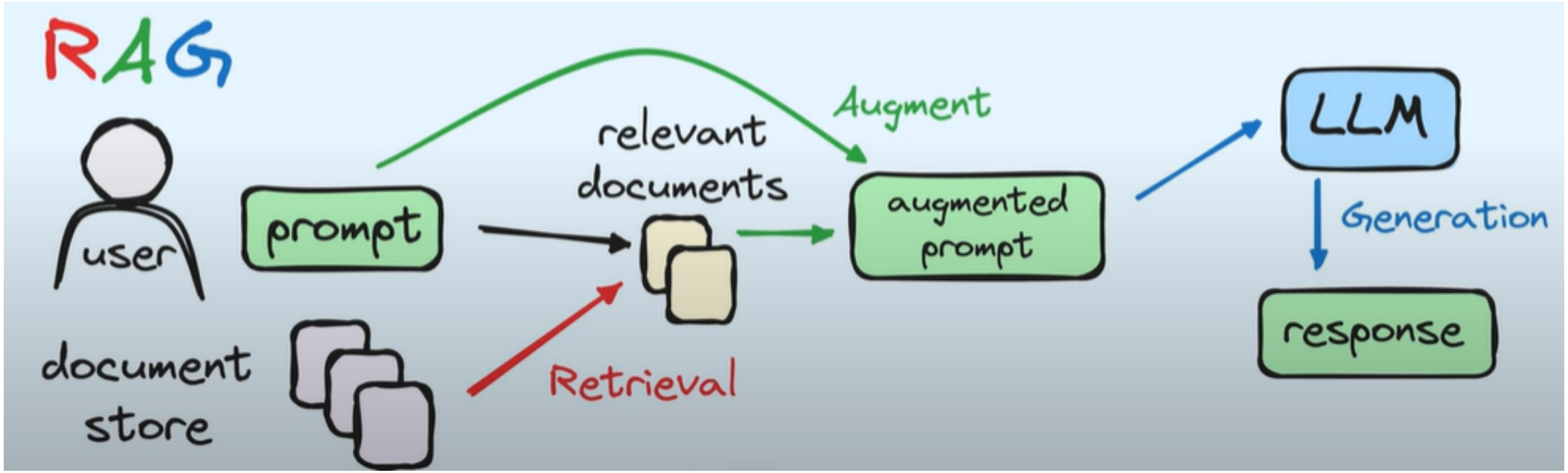
Non RAG Communication



RAG Setup


user
prompt
Course Catalog
physics 201 ...
kinematic equations
math 105
.... covers matrices
... english 205 ...
modern literature
Course 1
Course 2
...
Course N
document
store

RAG Communication



Advanced Retrieval Techniques

- Prefetching for Enhanced Performance
- Memoization: A Strategic Optimization Technique
- Concurrent Data Fetching: Enhancing Efficiency in Data Retrieval
- Lazy Loading: Enhancing Efficiency in Resource Loading

Demo

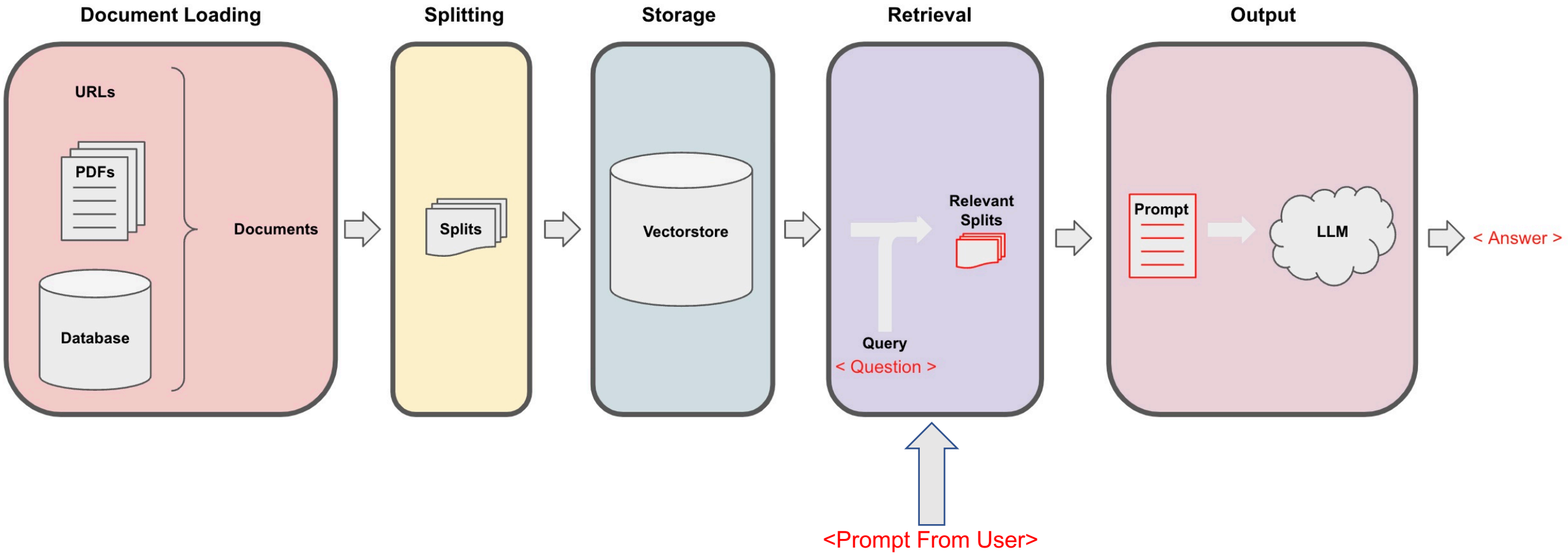
Case Study: Bring Your Own Data - Chat With PDF

Retrieval Augmented Generation(RAG)

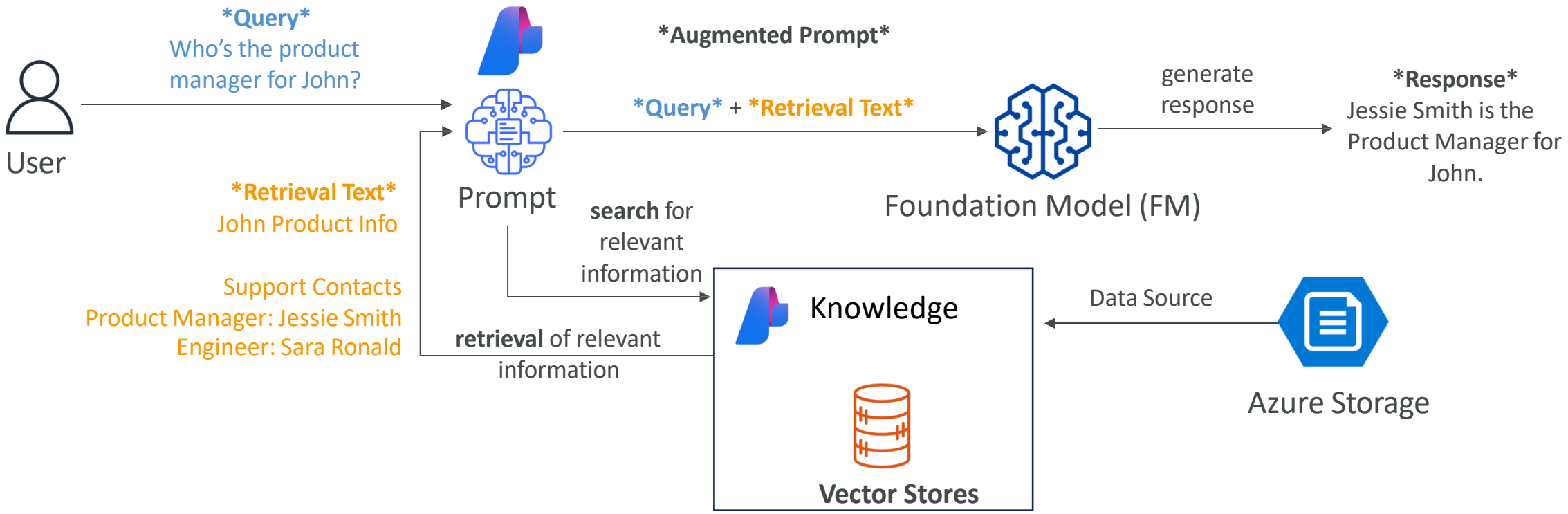
- In retrieval augmented generation (RAG), an LLM retrieves contextual documents from an external dataset as part of its execution.

- This is useful if we want to ask question about specific documents (e.g., our PDFs, a set of videos, etc).

Retrieval Augmented Generation(RAG)



Azure AI Foundry: RAG & Knowledge



Try Your Self

Lab: 3016 – 03 – Create a GAI App uses your own data

Thank You