Bringing Your Data to Life Conversational AI with LangChain

Parteek Kumar

Chat with your data using LangChain

- The ability to engage directly with data through conversational AI is a game-changer.
- Imagine querying complex datasets, documents, or databases as if you were having a conversation with a knowledgeable friend.
- LangChain, a powerful tool in the AI ecosystem, brings this vision to life.

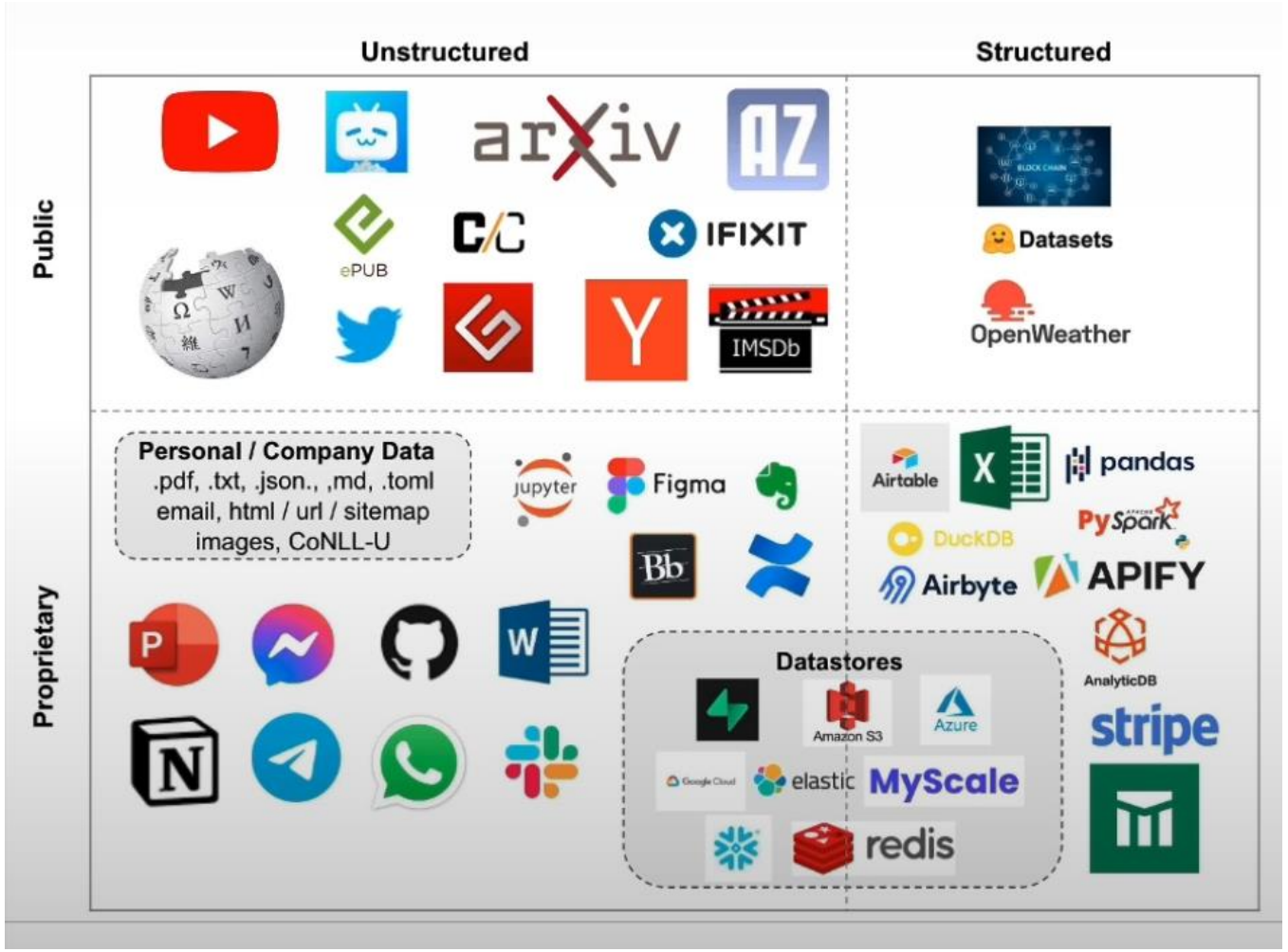
What is LangChain?

- LangChain is a robust library designed for building applications that leverage large language models (LLMs) to interact with data in a structured, conversational manner.
- Whether you're querying databases, analyzing documents, or automating workflows, LangChain can streamline these processes, allowing users to interact with data in natural language, rather than complex code or SQL queries.
- It's ideal for developers looking to enhance their applications with interactive AI capabilities.

Connect Your Data Source

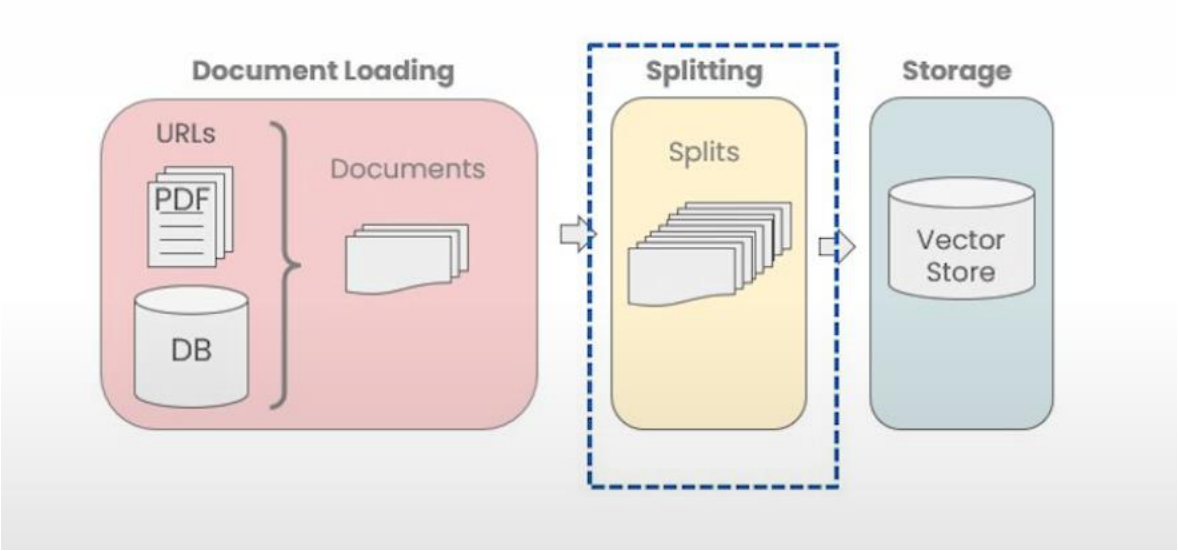
- LangChain allows you to connect to various data sources, including databases and document repositories.
- A document loader in LangChain is a component that imports, parses, and structures documents into a format suitable for indexing and querying.
- Once a document is loaded, it can be used by language models to provide answers based on the document's content.
- Document loaders support multiple file types, making LangChain versatile for projects involving diverse data formats.

Document Loaders



Document splitting

- By dividing extensive documents into smaller, manageable chunks, LangChain improves both query accuracy and processing efficiency.



Challenges of Document Splitting

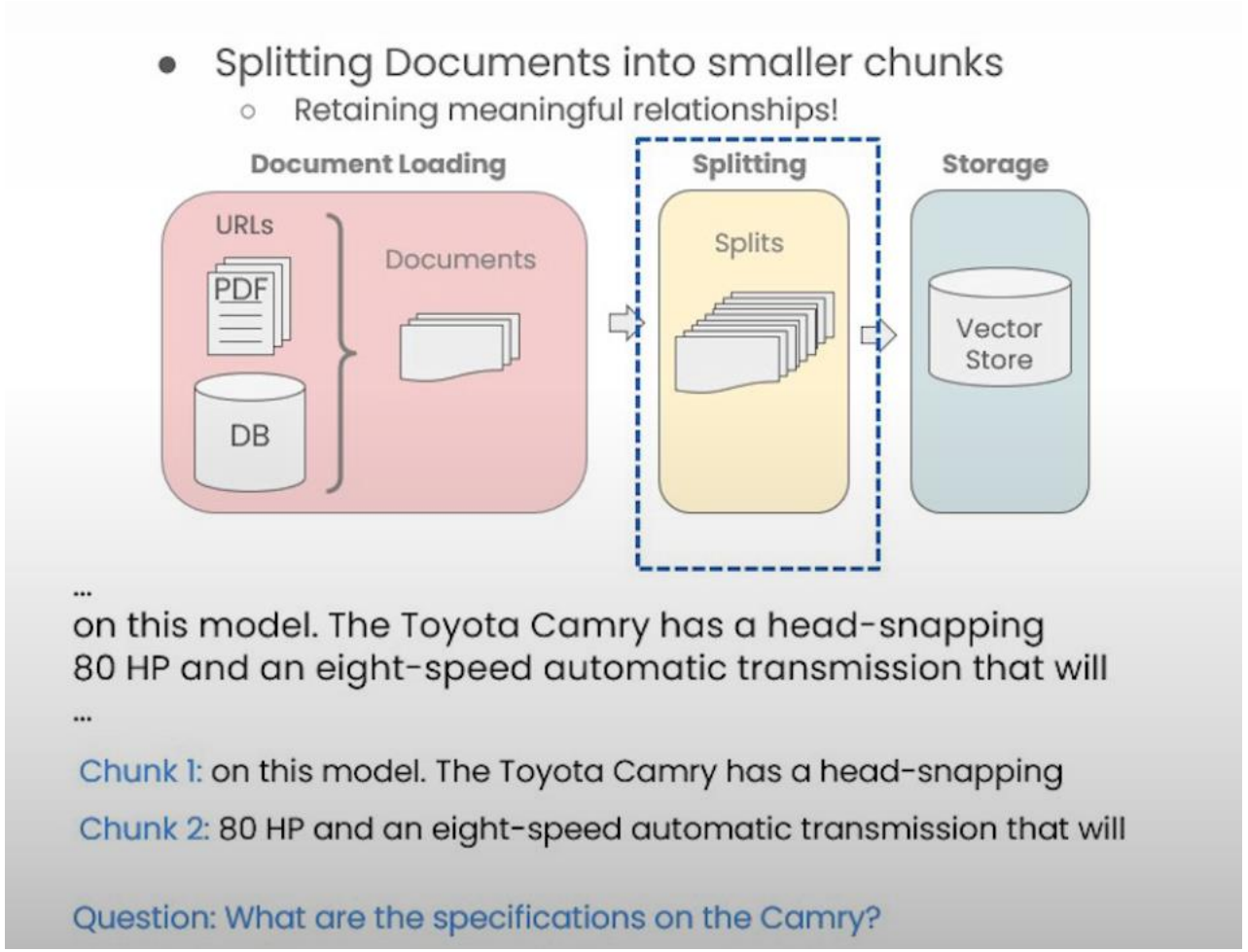
- "The Toyota Camry offers impressive fuel efficiency and advanced safety features on this model. The Toyota Camry has a head-snapping 80 HP and an eight-speed automatic transmission that will provide a smooth driving experience.“
- **Equal-Sized Chunks**
- **Chunk 1**: "The Toyota Camry offers impressive fuel efficiency and advanced"
- **Chunk 2**: "safety features on this model. The Toyota Camry has a"
- **Chunk 3**: "head-snapping 80 HP and an eight-speed automatic transmission"
- **Chunk 4**: "that will provide a smooth driving experience."

Challenges of Document Splitting

- **Equal-Sized Chunks**
- **Chunk 1**: "The Toyota Camry offers impressive fuel efficiency and advanced"
- **Chunk 2**: "safety features on this model. The Toyota Camry has a"
- **Chunk 3**: "head-snapping 80 HP and an eight-speed automatic transmission"
- **Chunk 4**: "that will provide a smooth driving experience.“
- If a user asks, "What are the specifications on the Camry?", the model faces a challenge as the full answer is divided across these multiple chunks.

Challenges of Document Splitting

- If a user asks, "What are the specifications on the Camry?", the model faces a challenge as the full answer is divided across these two chunks.
- Without any additional processing, the AI may only access one chunk at a time, resulting in incomplete or fragmented answers.



Solution: Overlap Between Chunks

- To overcome this challenge, LangChain uses a technique called **chunk overlap**. This approach involves adding a small overlap of text from the end of one chunk to the beginning of the next.
- This way, critical context that might be split between two chunks is preserved, enabling the model to access the full answer without losing coherence.
- For example, if we set a chunk overlap of around 50 characters, the overlapping portion of text would bridge **Chunk 1** and **Chunk 2**, ensuring that both chunks contain enough context to answer the question accurately.

LangChain DeepLearning.AI

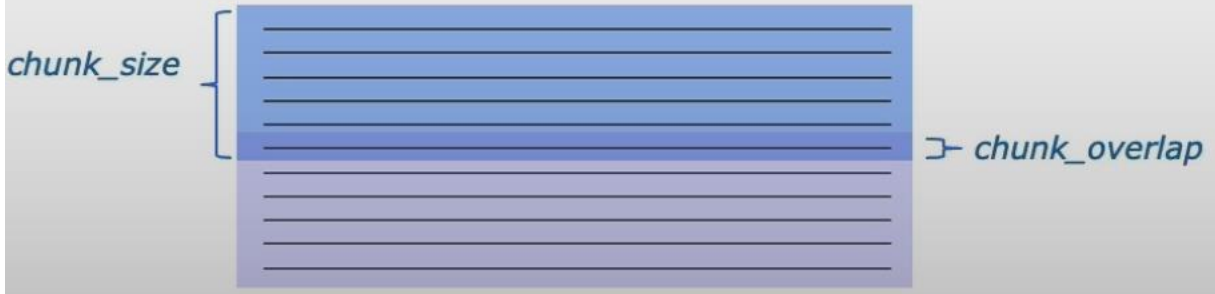
Example Splitter

```
langchain.text_splitter.CharacterTextSplitter(
    separator: str = "\n\n"
    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.



Types of 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 at tokens.
- **SentenceTransformersTokenTextSplitter**() - Implementation of splitting text that looks at tokens.
- ***RecursiveCharacterTextSplitter()*** **- Implementation of splitting text that looks at characters. Recursively tries to split by different characters to find one that works.**
- **Language()** – for CPP, Python, Ruby, Markdown etc
- **NLTKTextSplitter**() - Implementation of splitting text that looks at sentences using NLTK (Natural Language Tool Kit)
- **SpacyTextSplitter() -** Implementation of splitting text that looks at sentences using Spacy

Types of Splitters

- Each splitter has a specific method of breaking down text, depending on the nature of the document and the requirements o
- **CharacterTextSplitter()**
- Splits text at the character level.
- Useful for applications requiring precise control over character count in each chunk.
- **RecursiveCharacterTextSplitter()**
- Splits text by characters but does so recursively, trying different character counts to find a suitable split size.
- This approach ensures flexibility and adaptability in finding the best chunk size, making it suitable for documents of varying lengths and complexities

Types of Splitters

- r_splitter = RecursiveCharacterTextSplitter(
  chunk_size=150, chunk_overlap=0,
  separators=["\n\n", "\n", "\. ", " ", ""])
- r_splitter.split_text(some_text)

LangChain DeepLearning.AI

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.

Embeddings

- Embeddings represent data (like text or documents) as numerical vectors, capturing the meaning and relationships between words or phrases.
- Embeddings capture semantic information by representing words, sentences, or entire documents as vectors in a high-dimensional space.
- In this space, **similar meanings or contexts are represented by vectors that are close to each other, while unrelated meanings are further apart.**

Embeddings in Action

- Suppose we have three sentences:

1. "The cat is sleeping on the couch."
2. "The dog is resting on the sofa."
3. "The car is parked in the garage."

- When we generate embeddings for these sentences, sentences with similar meanings will have vectors close to each other.

Embeddings in Action

1."The cat is sleeping on the couch."
2."The dog is resting on the sofa."
3."The car is parked in the garage."

Synonyms and Similar Phrases

- The words "couch" and "sofa" are synonyms, and "sleeping" and "resting" convey similar actions.
- Even though the words are different, an embedding model captures the similarity in meaning by placing the vectors for sentences 1 and 2 closer to each other.
- This happens because embeddings capture not only individual words but also the relationship between words based on their context in large datasets used for training.

Embeddings in Action

1."The cat is sleeping on the couch."
2."The dog is resting on the sofa."
3."The car is parked in the garage."

Synonyms and Similar Phrases

- Contextual Differences

Sentence 3 is about a car in a garage, which is a very different concept from pets resting on furniture. Therefore, the embedding for sentence 3 will be farther away from sentences 1 and 2, reflecting the difference in context.

Embeddings in Action

1."The cat is sleeping on the couch."
2."The dog is resting on the sofa."
3."The car is parked in the garage."

- Let's say you ask a question: "Where is the dog lying down?"
- When converted to an embedding, this query will generate a vector close to the embeddings for sentences 1 and 2, because "lying down" is semantically close to "resting" and "sleeping," and "dog" is semantically close to "cat" (both are animals). The model will retrieve sentences 1 and 2 as the most relevant responses, even though they don't contain the exact words from the query.

LangChain DeepLearning.AI

Embeddings

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

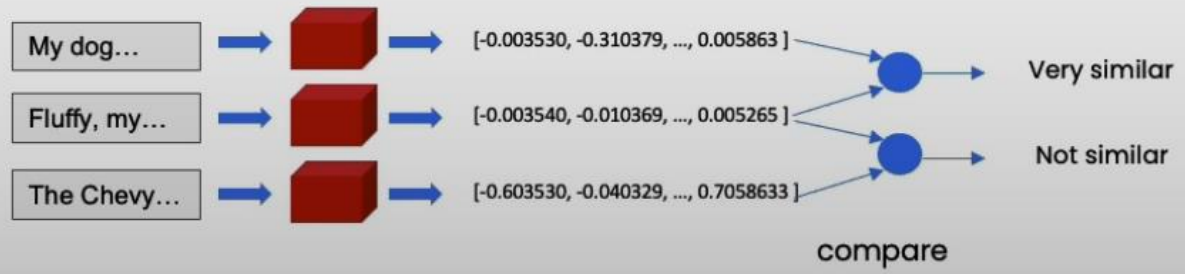
1) My dog Rover likes to chase squirrels.
2) Fluffy, my cat, refuses to eat from a can.
3) The Chevy Bolt accelerates to 60 mph in 6.7 seconds.

LangChain DeepLearning.AI

Embeddings

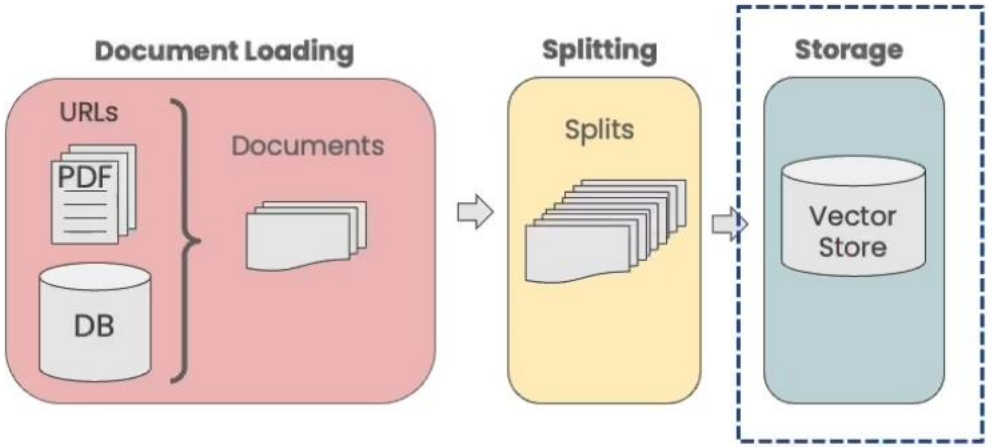
- Embedding vector captures content/meaning
- Text with similar content will have similar vectors

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.



Vector Stores in LangChain
The Backbone of Efficient Data Retrieval

- A vector store is a database specifically designed to store, index, and retrieve vector representations (embeddings) of data.
- These vector representations are numerical encodings of text that capture semantic meaning, allowing the language model to identify relevant pieces of information based on similarity rather than exact keyword matches.



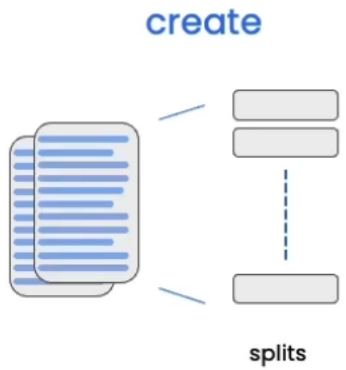
LangChain

DeepLearning.AI

Vector Store

create

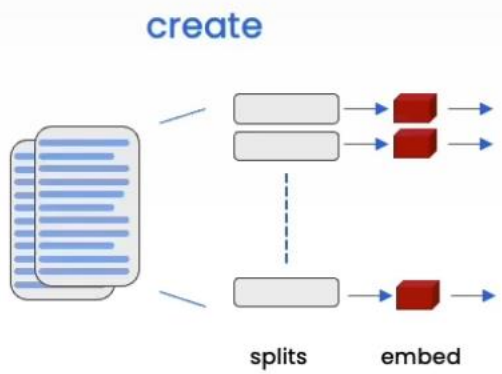
Vector Store

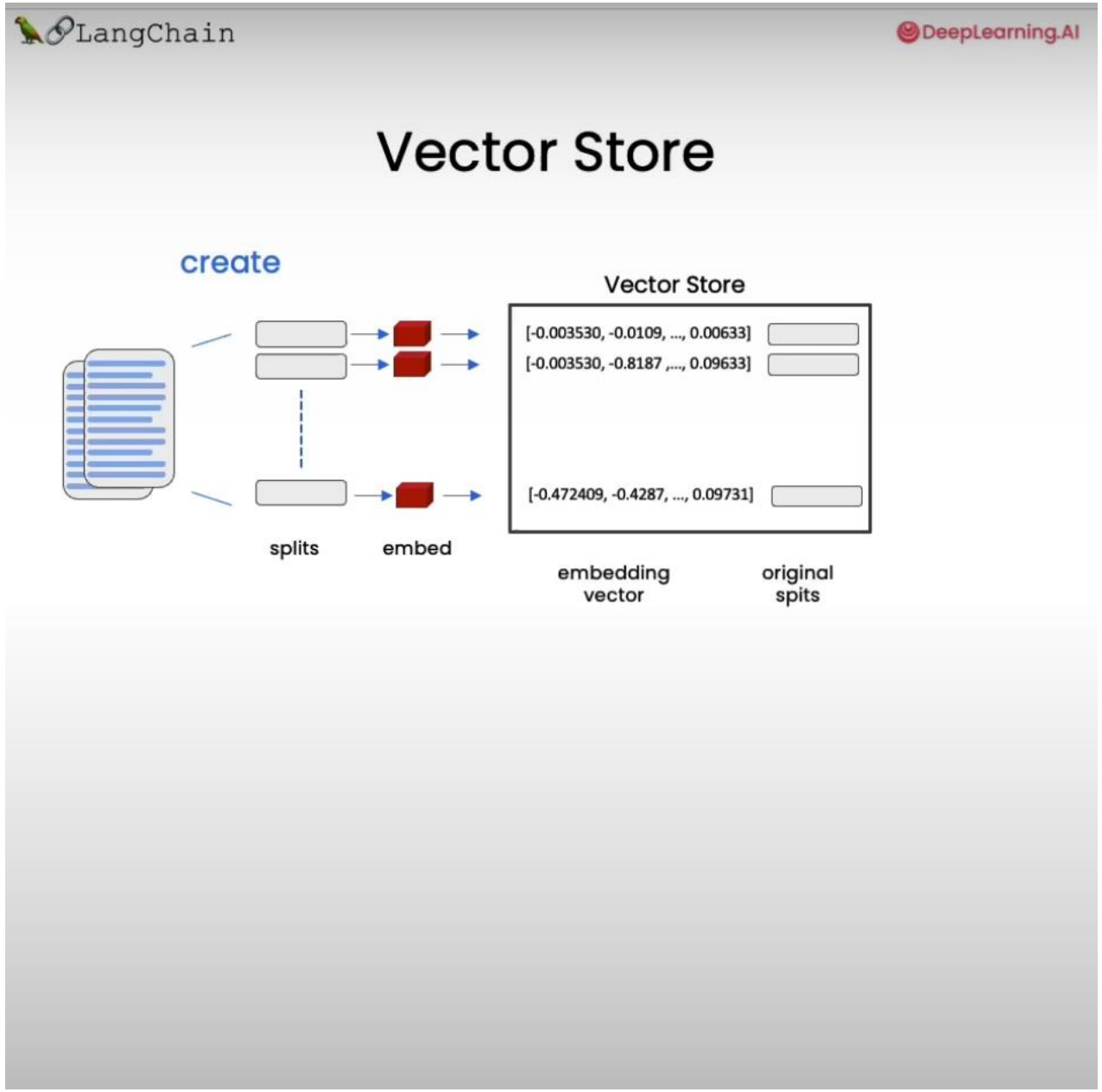


LangChain

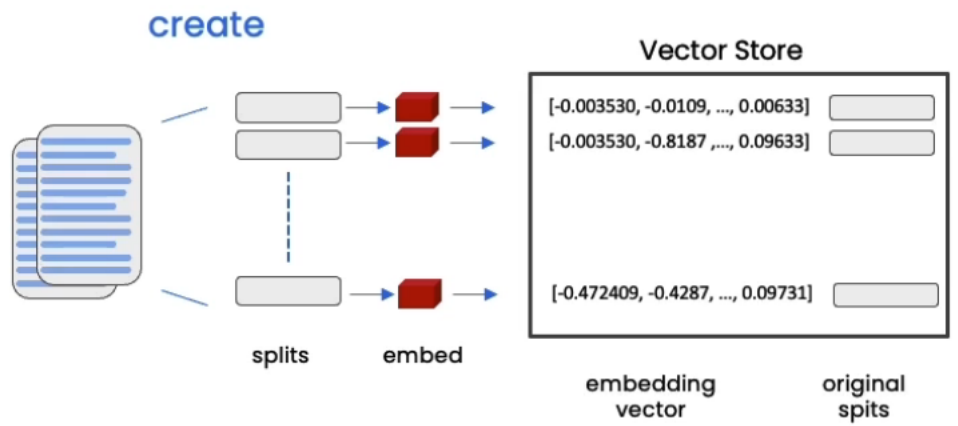
DeepLearning.AI

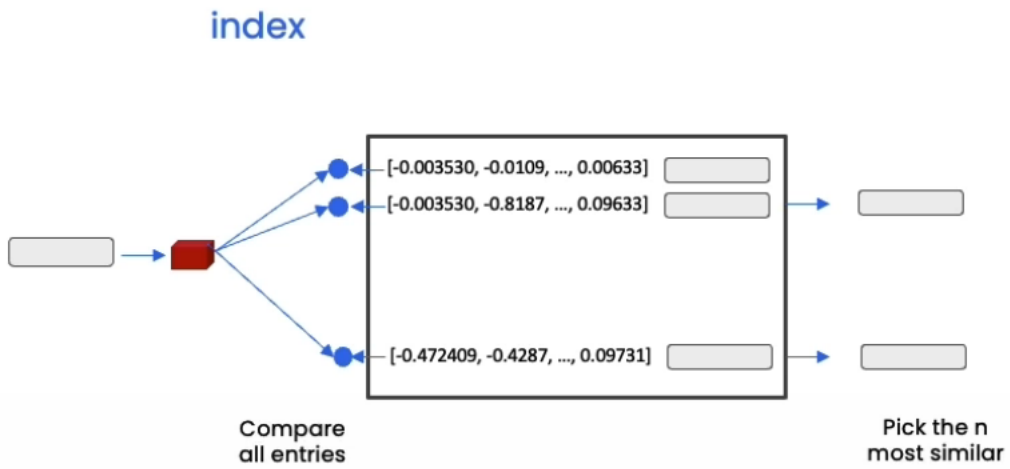
Vector Store




LangChain
DeepLearning.AI
Vector Store
create
Vector Store
[-0.003530, -0.0109, ..., 0.00633]
[-0.003530, -0.8187 ,..., 0.09633]
[-0.472409, -0.4287, ..., 0.09731]
splits
embed
embedding vector
original spits

Vector Store

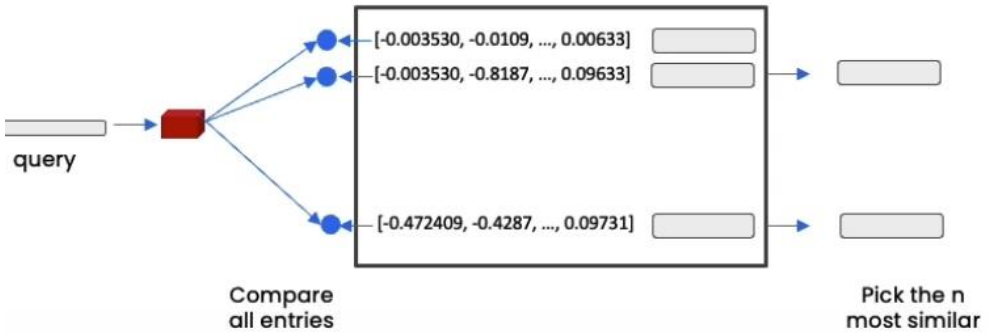




LangChain DeepLearning.AI

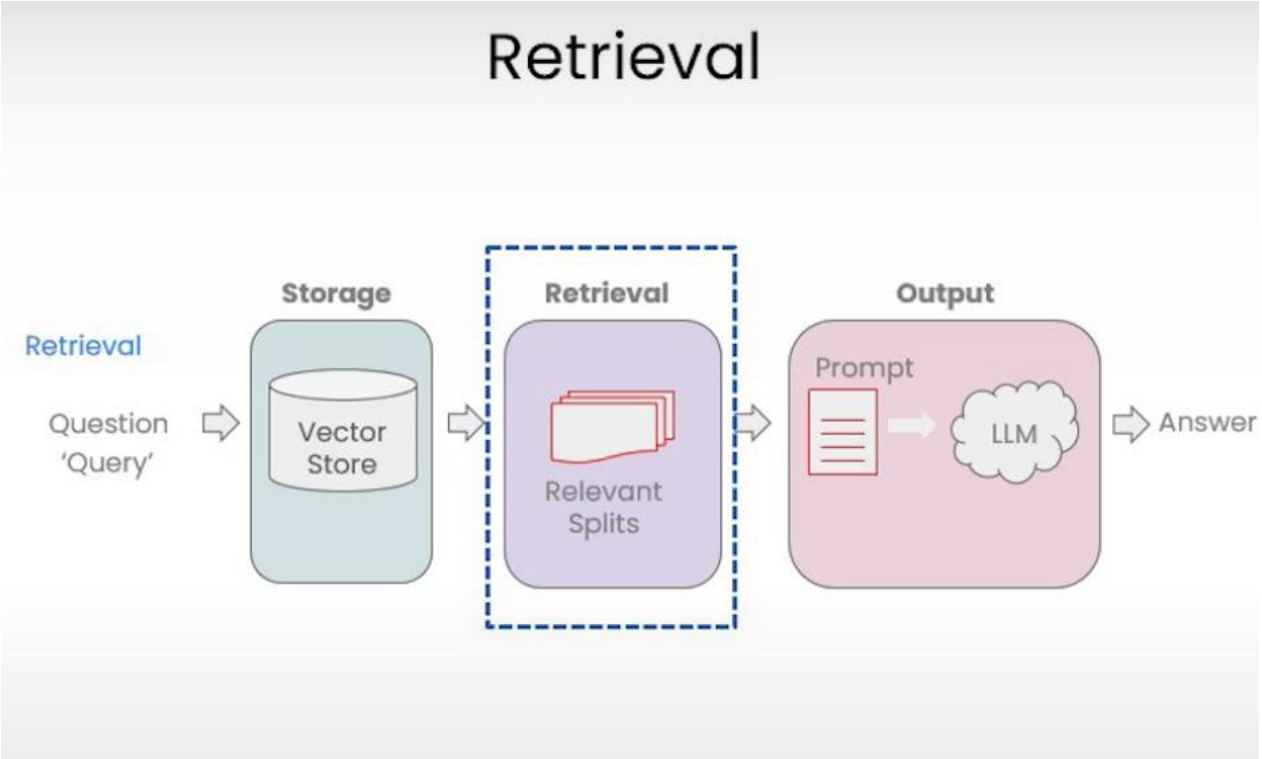
Vector Store/Database

index



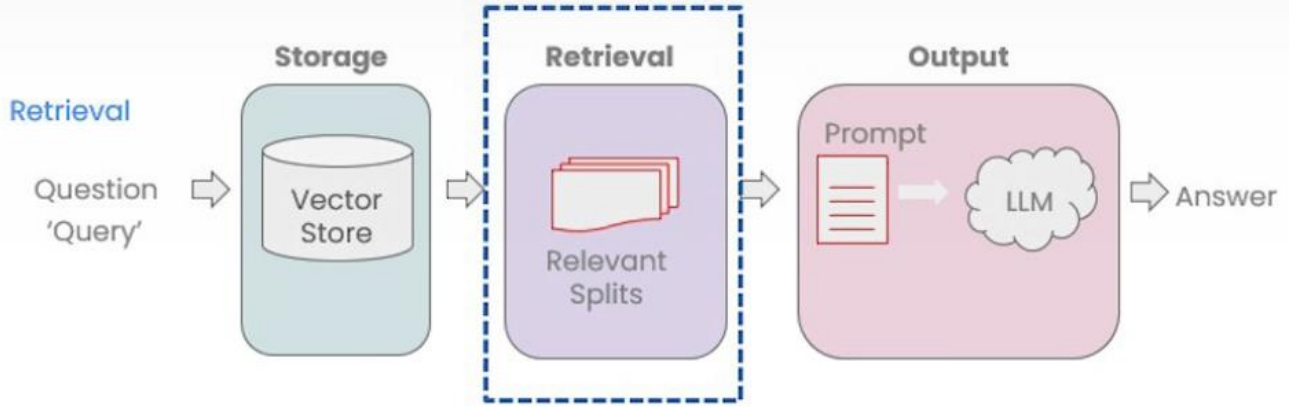
Process with llm

The returned values can now fit in the LLM context


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

LangChain DeepLearning.AI

Retrieval



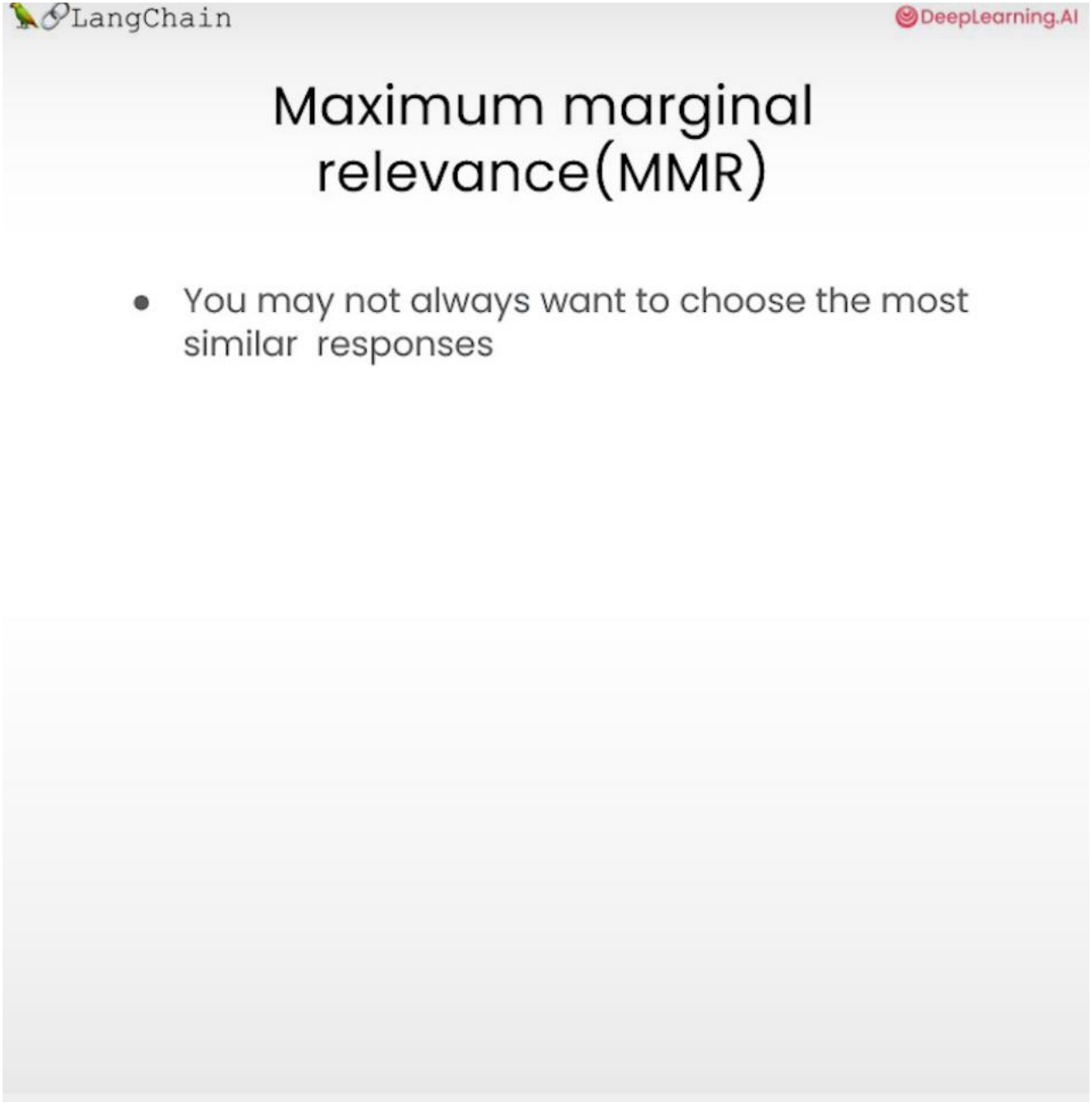
- Accessing/indexing the data in the vector store
  - Basic semantic similarity
  - Maximum marginal relevance
  - Including Metadata
- LLM Aided Retrieval

Retrieval
Maximum Marginal Relevance (MMR)

- **Maximum Marginal Relevance (MMR)** is an algorithm designed to retrieve results **that are not only relevant but also diverse,** reducing redundancy in responses.
- **Maximum Marginal Relevance (MMR)** is a technique used to select diverse responses **that balance relevance and novelty,** which can be especially helpful when you don't necessarily want the system to return only the most similar responses.

How MMR Works

- MMR evaluates potential responses by considering two main factors:
- **Relevance**: How closely the response matches the user's query.
- **Diversity (Marginal Relevance)**: How much new information the response brings compared to what has already been provided.
- MMR calculates a score for each response by maximizing relevance while minimizing similarity to previously selected responses.


LangChain
DeepLearning.AI
Maximum marginal relevance(MMR)
You may not always want to choose the most similar responses

LangChain DeepLearning.AI

Maximum marginal relevance(MMR)

- You may not always want to choose the most similar responses

Tell me about all-white mushrooms with large fruiting bodies

The Amanita phalloides has a large and imposing epigeous (aboveground) fruiting body (basidiocarp).

A mushroom with a large fruiting body is the Amanita phalloides. Some varieties are all-white.

AA. phalloides, a.k.a Death Cap, is one of the most poisonous of all known mushrooms.

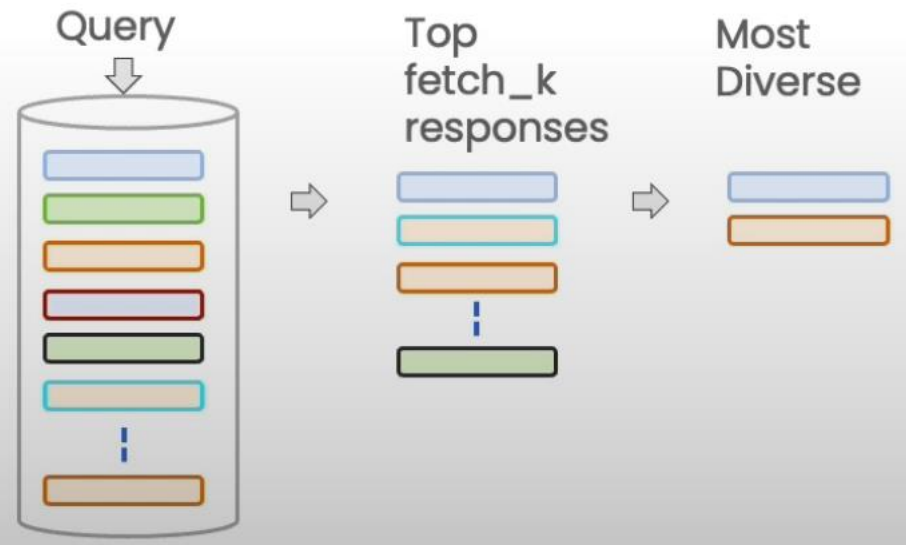
Most Similar

MMR

LangChain DeepLearning.AI

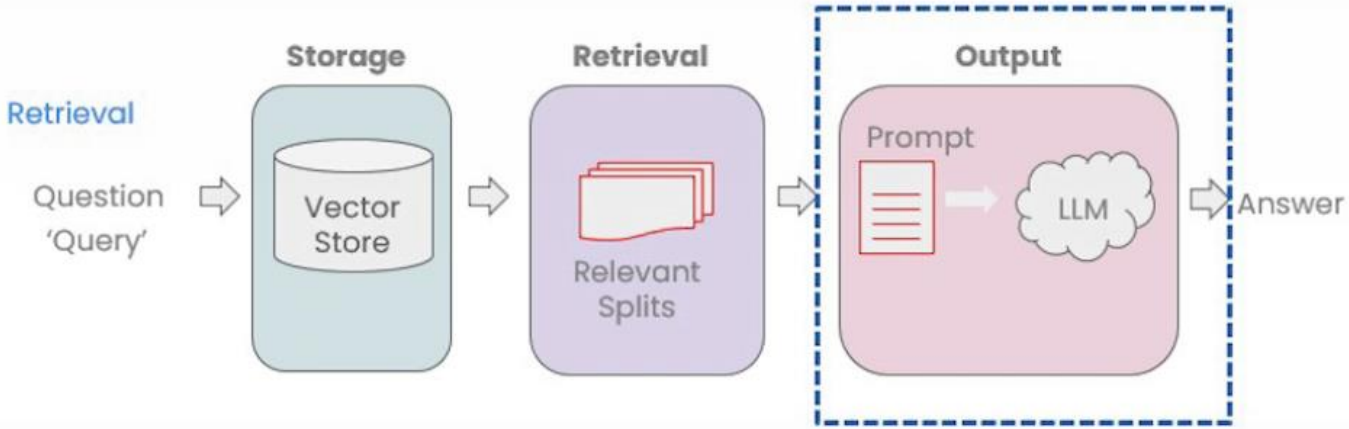
MMR algorithm

- Query the Vector Store
- Choose the `fetch_k` most similar responses
- Within those responses choose the `k` most diverse



LangChain DeepLearning.AI

Question Answering



- Multiple relevant documents have been retrieved from the vector store
- Potentially compress the relevant splits to fit into the LLM context
- Send the information along with our question to an LLM to select and format an answer

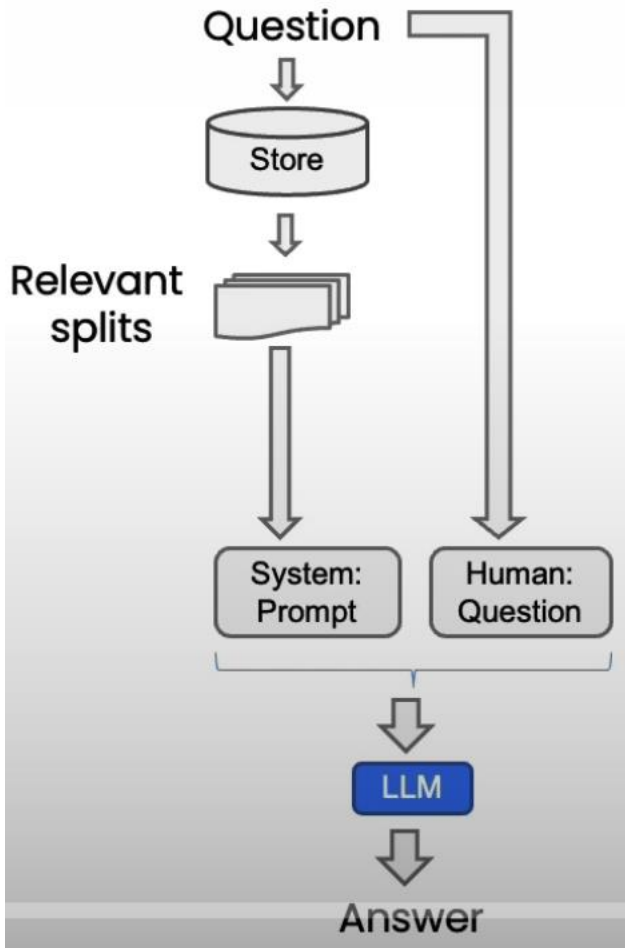
Retrieval-Augmented Generation

- **Retrieval-Augmented Generation**, a hybrid technique in artificial intelligence that combines retrieval-based systems with generative AI models.
- RAG systems are designed to improve the accuracy, relevance, and informativeness of responses by integrating external knowledge retrieval into the generation proces

LangChain DeepLearning.AI

RetrievalQA chain

```
RetrievalQA.from_chain_type(, chain_type="stuff",...)
```



Question is applied to the Vector Store as a query

Vector store provides k relevant documents

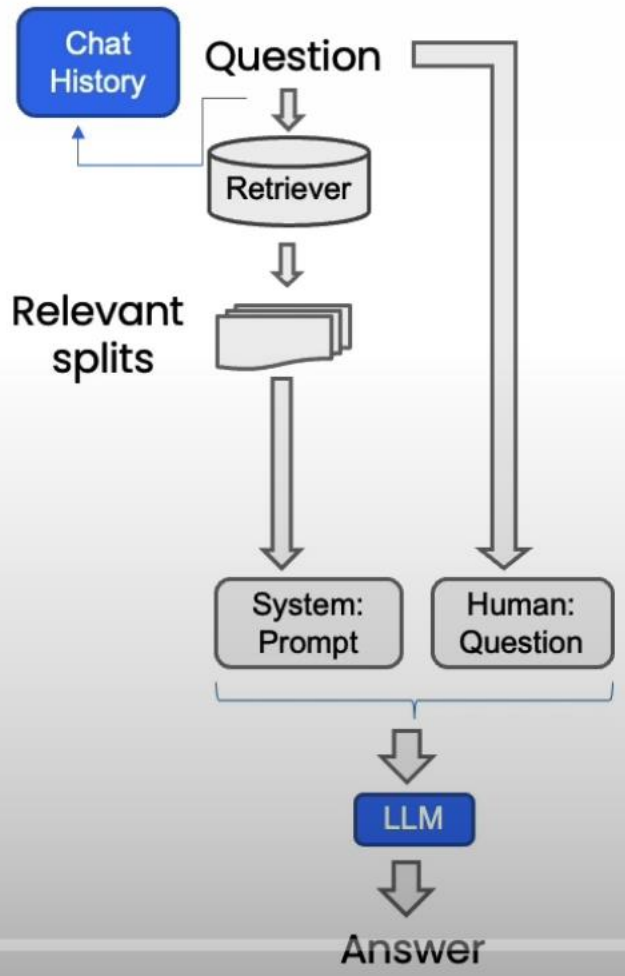
Docs and original question are sent to an LLM

LangChain

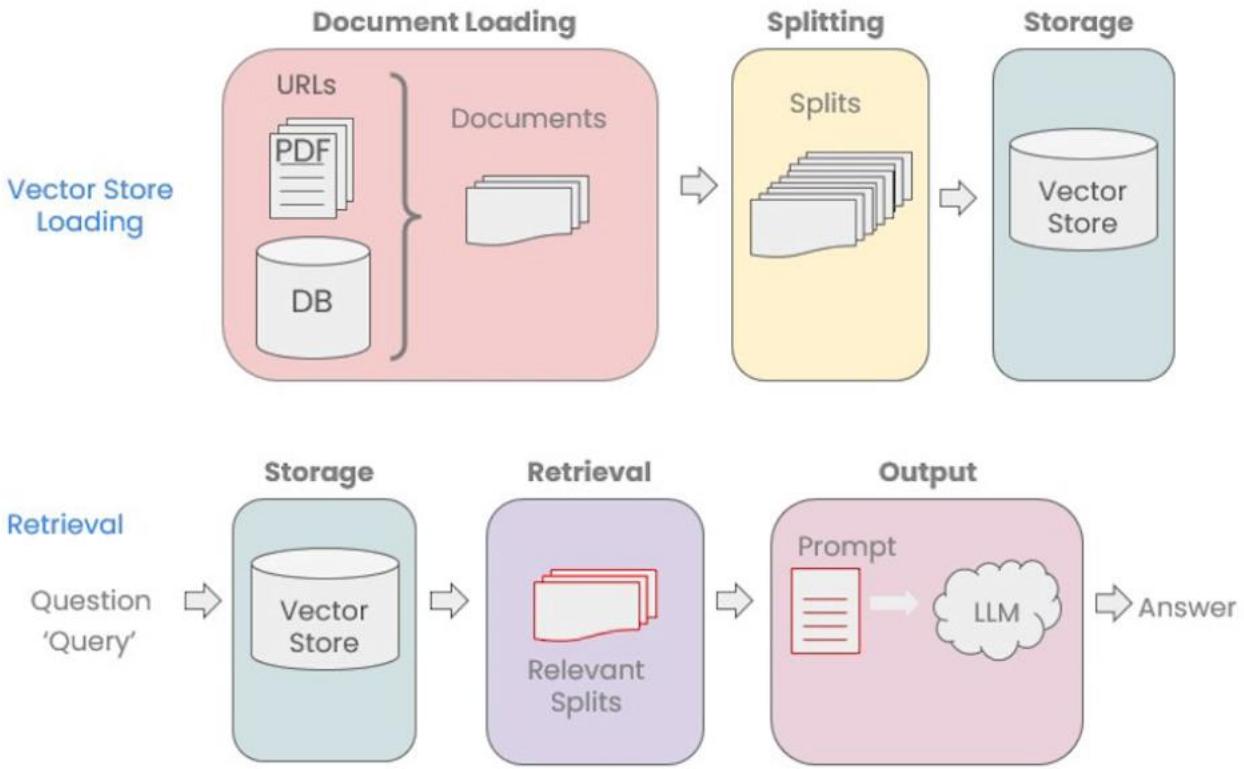
DeepLearning.AI

ConversationalRetrievalChain

```
qa = ConversationalRetrievalChain.from_llm(ChatOpenAI(temperature=0),
vectorstore.as_retriever(), memory=memory)
```



LangChain
Chat with Your Data



Academic Contributions

Online Udemy Courses: 07

- Number of Learners: 43000+
- Number of Countries: 160+
- Average Rating: 4.4

YouTube Channel:: Simplifying Computer Education

- Subscribers: 7000+
- Videos: 500+
- Views: 6,35,000 +
- Watch Hours: 31000 +

FDP and Workshops Conducted: 60+

Machine learning has become a dominant problem-solving technique in the modern world, with applications ranging from search engines and social media to self-driving cars and artificial intelligence. This lucid textbook presents the theoretical foundations of machine learning algorithms, and then illustrates each concept with its detailed implementation in Python to allow beginners to effectively implement the principles in real-world applications. All major techniques, such as regression, classification, clustering, deep learning, and association mining, have been illustrated using step-by-step coding instructions to help inculcate a "learning by doing" approach. The book has no prerequisites, and covers the subject from the ground up, including a detailed introductory chapter on the Python language. As such, it is going to be a valuable resource not only for students of computer science, but also for anyone looking for a foundation in the subject, as well as professionals looking for a ready reckoner.

Features

- Algorithms are explained in detail with examples using a step-by-step approach to make learning easy and simple.
- Ample solved examples and practice problems help illustrate the material further.
- Each chapter ends with brief summaries, main takeaways further reading and resources on the Internet.
- GitHub resources that provide access to datasets, sample code, and examples have been included.
- Advanced topics like deep learning, convolutional neural network, and recurrent neural network have been covered extensively.
- An online supplements package includes a solutions manual and lecture slides for instructors, and further online reading and a chapter-wise list of project ideas for students.
- Main topics have also been illustrated with video resources from the author.

Parteek Bhatia is Professor in the Department of Computer Science and Engineering at Thapar Institute of Engineering and Technology, Patiala, India, an institution he has been associated with since 2004. His research interests include machine learning, natural language processing, and explainable AI. He has authored several textbooks in the domain of databases, including *Data Mining and Data Warehousing: Principles and Practical Techniques* published by Cambridge University Press in 2019.

Online Resources
www.cambridge.org/9781009170246

Cover image source: Getty Images/Westend61

Machine Learning *with* Python

Bhatia

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-009-17024-6

9 781009 170246

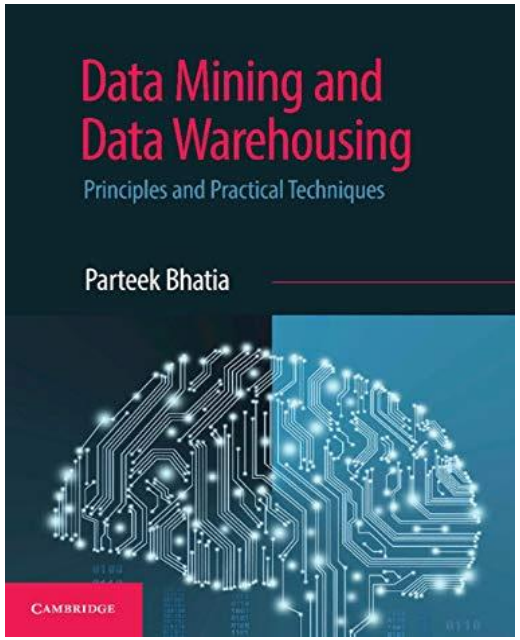
CAMBRIDGE

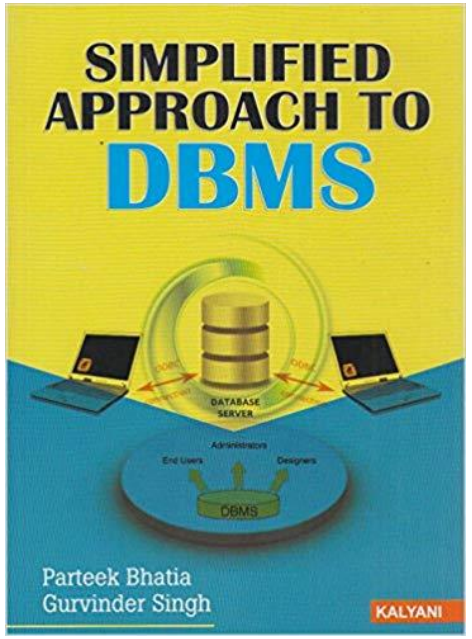
Machine Learning *with* Python

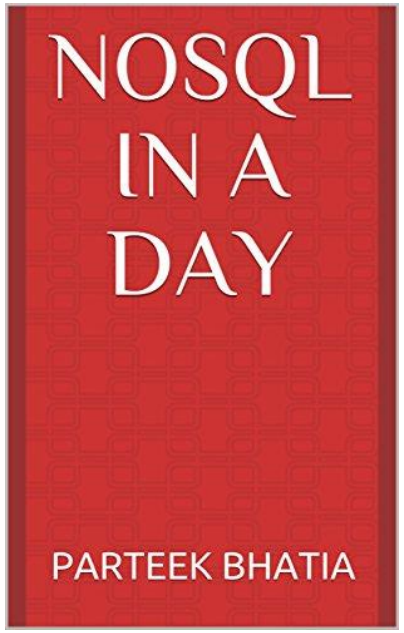
Principles and Practical Techniques

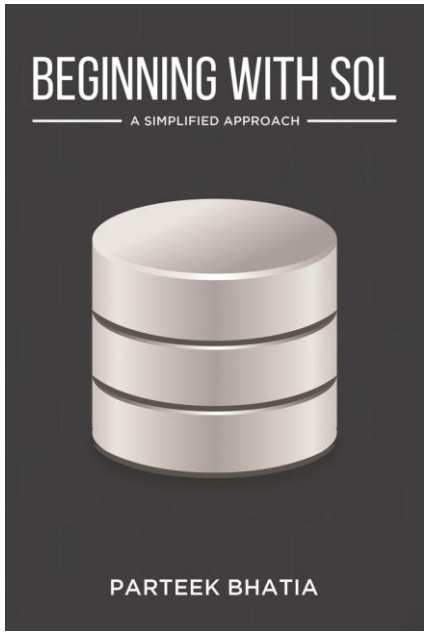
Parteek Bhatia

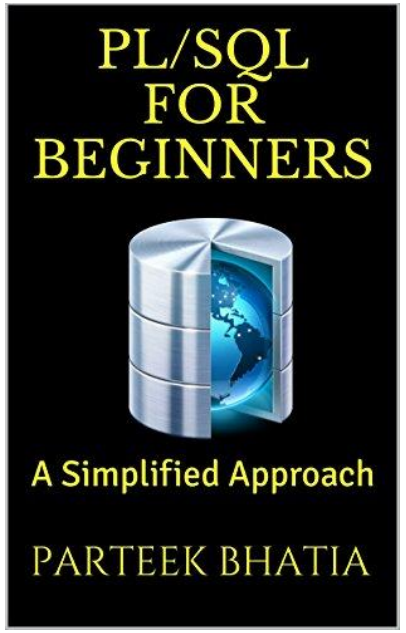
Other Books





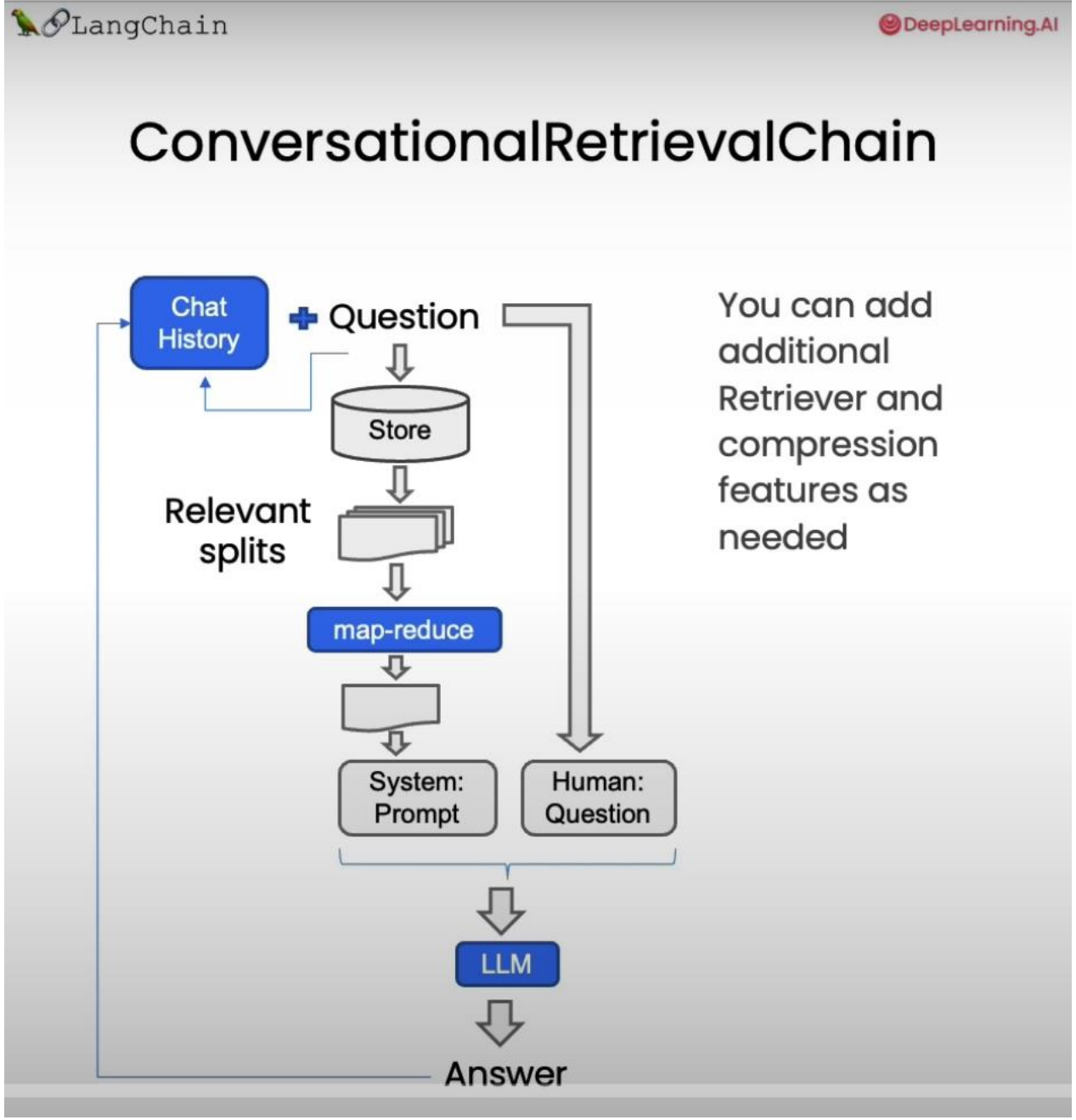






For more information visit: www.parteekbhatia.com

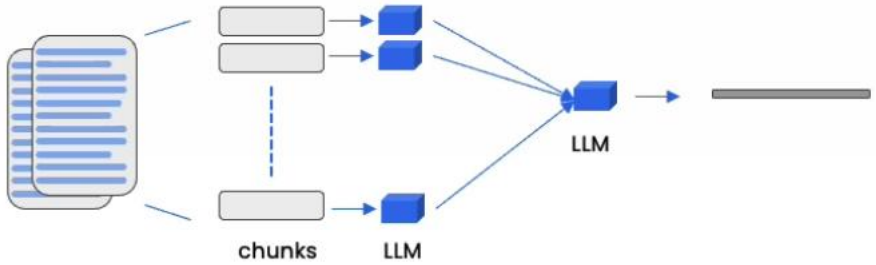
Thanks


LangChain
DeepLearning.AI
ConversationalRetrievalChain
Chat History
Question
Store
Relevant splits
map-reduce
System: Prompt
Human: Question
LLM
Answer
You can add additional Retriever and compression features as needed

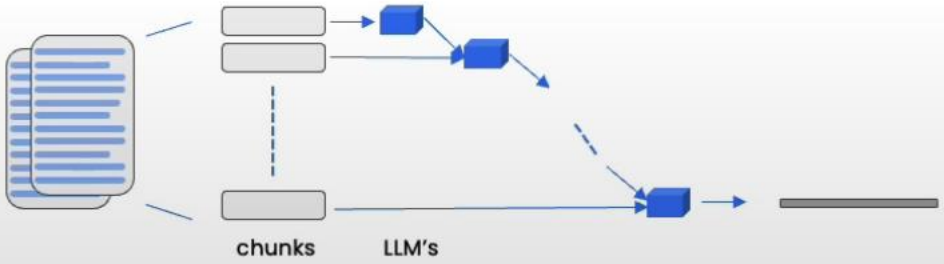
LangChain DeepLearning.AI

3 additional methods

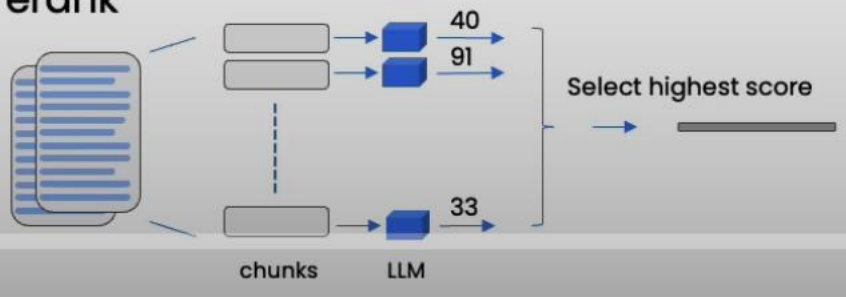
1. Map_reduce



2. Refine



3. Map_rerank



LangChain DeepLearning.AI

LLM Aided Retrieval

- There are several situations where the Query applied to the DB is more than just the Question asked.
- One is SelfQuery, where we use an LLM to convert the user question into a query