**A diagram of a structure

Description automatically generatedLANGCHAIN**

LangChain offers a variety of modules that can be used to create language model applications. These modules can be combined to create more complex applications or can be used individually for simpler applications.

A logo with text on it

Description automatically generated

* **Models** are the building block of LangChain providing an interface to different types of AI models. Large Language Models (LLMs), Chat and Text Embeddings models are supported model types.
* **Prompts** refers to the input to the model, which is typically constructed from multiple components. LangChain provides interfaces to construct and work with prompts easily - Prompt Templates, Example Selectors and Output Parsers.
* **Memory** provides a construct for storing and retrieving messages during a conversation which can be either short term or long term.
* **Indexes** help LLMs interact with documents by providing a way to structure them. LangChain provides Document Loaders to load documents, Text Splitters to split documents into smaller chunks, Vector Stores to store documents as embeddings, and Retrievers to fetch relevant documents.
* **Chains** let you combine modular components (or other chains) in a specific order to complete a task.
* **Agents** are a powerful construct in LangChain allowing LLMs to communicate with external systems via Tools and observe and decide on the best course of action to complete a given task.

**Documents: -**

Document in LangChain refers to an unstructured text consisting of page\_content referring to the content of the data and metadata (data describing attributes of page content).

## Models - The interface to the AI brains

LangChain supports : -

* LLMs
* Chat Models
* Text Embedding Models

**LLMs: -**

The third type of models are text embedding models. These models take text as input and return a list of floats/vectors.

## **Prompts: -**

Prompts are text used as instructions to your model.

### ****Prompt Template: -****

Prompt Template is an object that helps to create prompts based on a combination of user input, other non-static information and a fixed template string.

### ****Output Parsers****

Output Parsers help to format the output of a model. Usually used for structured output.

Two main ideas:

* **Format Instructions**: An autogenerated prompt that tells the LLM how to format its response based off desired result
* **Parser**: A method to extract model's text output into a desired structure (usually json)

## **Indexes**

Indexes refer to ways to structure documents for LLMs to work with them.

### ****Document Loaders: -****

Document loaders are ways to import data from other sources. See the [growing list](https://python.langchain.com/en/latest/modules/indexes/document_loaders.html) of document loaders here.

### ****Text Splitters: -****

Text Splitters are a way to deal with input token limits of LLMs by splitting text into chunks.

### ****Retrievers: -****

Retrievers are a way of storing data such that it can be queried by a language model. Easy way to combine documents with language models

**VectorStores:-**

Vector Store is a common type of index or a database to store vectors (numerical embeddings). Conceptually, think of them as tables with a column for embeddings (vectors) and a column for metadata.

* [Chroma](https://www.trychroma.com/) & [FAISS](https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/) are easy to work with locally

## **Memory: -**

Memory is the concept of storing and retrieving data in the process of a conversation. Memory helps LLMs remember information you've chatted about in the past or more complicated information retrieval.

There are many types of memory Example: -

### ConversationBufferMemory

### Memory keeps conversation state throughout a user’s interactions with a language model. ConversationBufferMemory memory allows for storing of messages and then extracts the messages in a variable.

## **Chains**: -

Chains are a generic concept in LangChain allowing to combine different LLM calls and action automatically.

Ex:

Summary #1, Summary #2, Summary #3 --> Final Summary

There are many types of Chains Example: -

### Simple Sequential Chains: -

### Sequential chains are a series of chains, called in deterministic order.  SimpleSequentialChain are easy chains where each step uses the output of an LLM as an input into another. Good for breaking up tasks (and keeping the LLM focused).

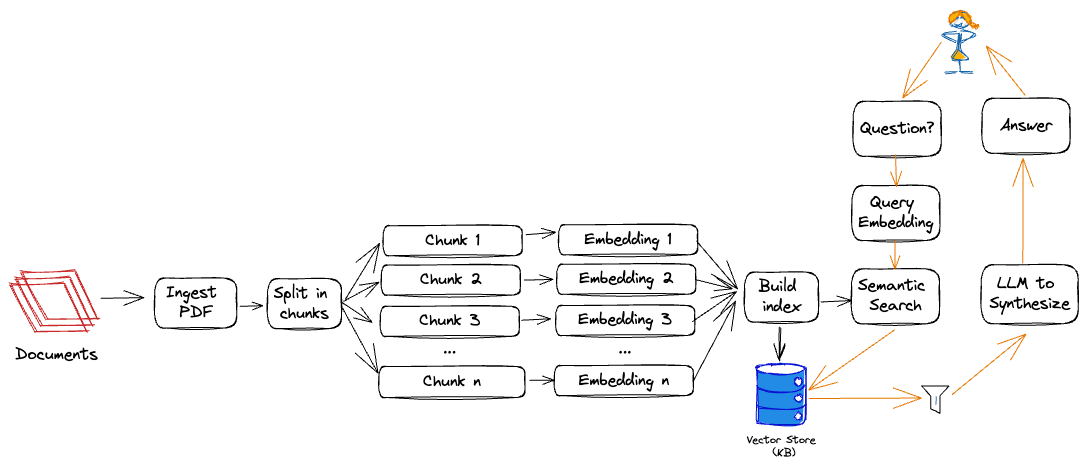
### Summarization Chain: -

Summarization Chain easily runs through a long numerous document and get a summary.

There are multiple chain types such as Stuffing, Map-Reduce, Refine.

### Question/Answering Chain: -

[Question Answering Chains] easily do QA over a set of documents using QA chain. There are multiple ways to do this with LangChain. We use **RetrievalQA** chain which uses **load\_qa\_chain** under the hood.



Different Steps for chain: -

1. User Input: The LLM chain starts by taking user input, which can be in the form of a question, a command, or any other text-based input. This input serves as the initial prompt for the LLM chain.
2. Prompt Template Integration: One of the essential components of an LLM chain is the prompt template. The chain utilizes a prompt template to format the user input in a specific structure that the LLM can understand.
3. Formatting and Preprocessing: After applying the prompt template, the chain may perform additional formatting and preprocessing steps to enhance the input for the subsequent LLM processing. These steps can include tokenization, normalization, or any other necessary transformations to optimize the data for the LLM.
4. Language Model Processing: The formatted and preprocessed prompt is then passed to the LLM component of the chain. the model capable of understanding and generating human-like text, processes the input prompt and generates a response.
5. Output Integration: The response generated by the LLM becomes the output of the current step in the chain. Depending on the specific requirements and objectives of the application, this output can be used in various ways. It can be displayed to the user, further processed, or fed into the next component in the chain.