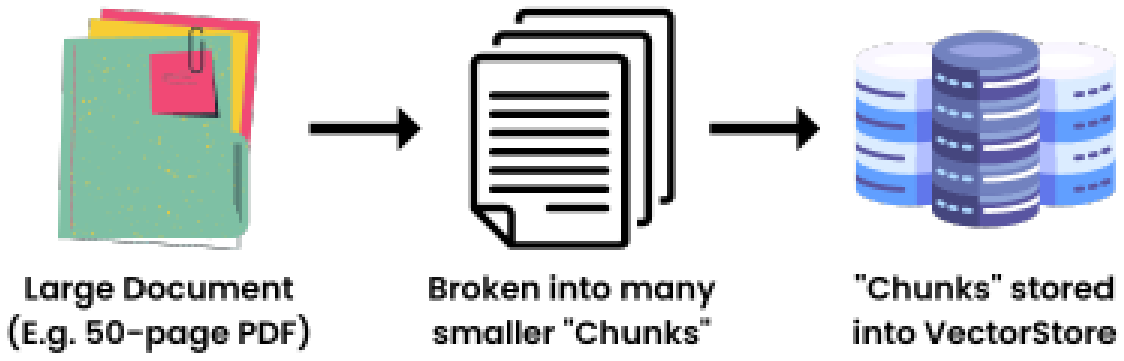
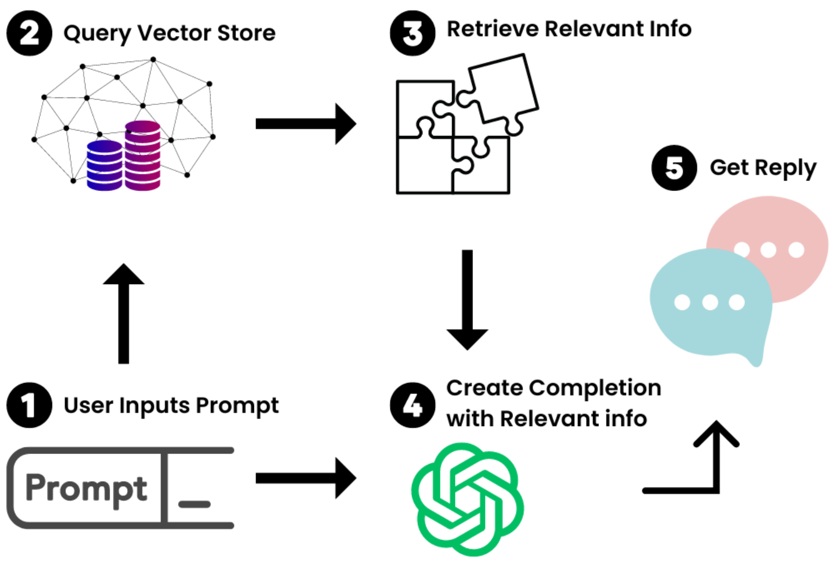
LangChain

Open source framework that allows AI developers to combine Large Language Models (LLMs) with external data.

LangChain just composes large amounts of data that can easily be referenced by a LLM with as little computation power as possible.


Large Document
(E.g. 50-page PDF)
Broken into many smaller "Chunks"
"Chunks" stored into VectorStore

When we insert a prompt into our new chatbot, LangChain will query the Vector Store for relevant information. Think of it as a mini-Google for your document. Once the relevant information is retrieved, we use that in conjunction with the prompt to feed to the LLM to generate our answer.



Pinecone is the Vector Store that we will be using in conjunction with LangChain.

A LangChain application consists of 5 main components:

- Models (LLM Wrappers)
- Prompts
- Chains
- Embeddings and Vector Stores
- Agents

Models (LLM Wrappers)

To interact with our LLMs, we are going to instantiate a wrapper for OpenAI's GPT models. In this case, we are going to use OpenAI's GPT-3.5-turbo, as it's the most cost-efficient. But if you have access, feel free to use the more powerful GPT4.

Prompts

LangChain provides PromptTemplates that allow you to dynamically change the prompts with user input, similar to how regex are used.

Chains

Chains allow you to take simple PromptTemplates and build functionality on top of them. Essentially, chains are like composite functions that allow you to integrate your PromptTemplates and LLMs together.

Embeddings and Vector Stores

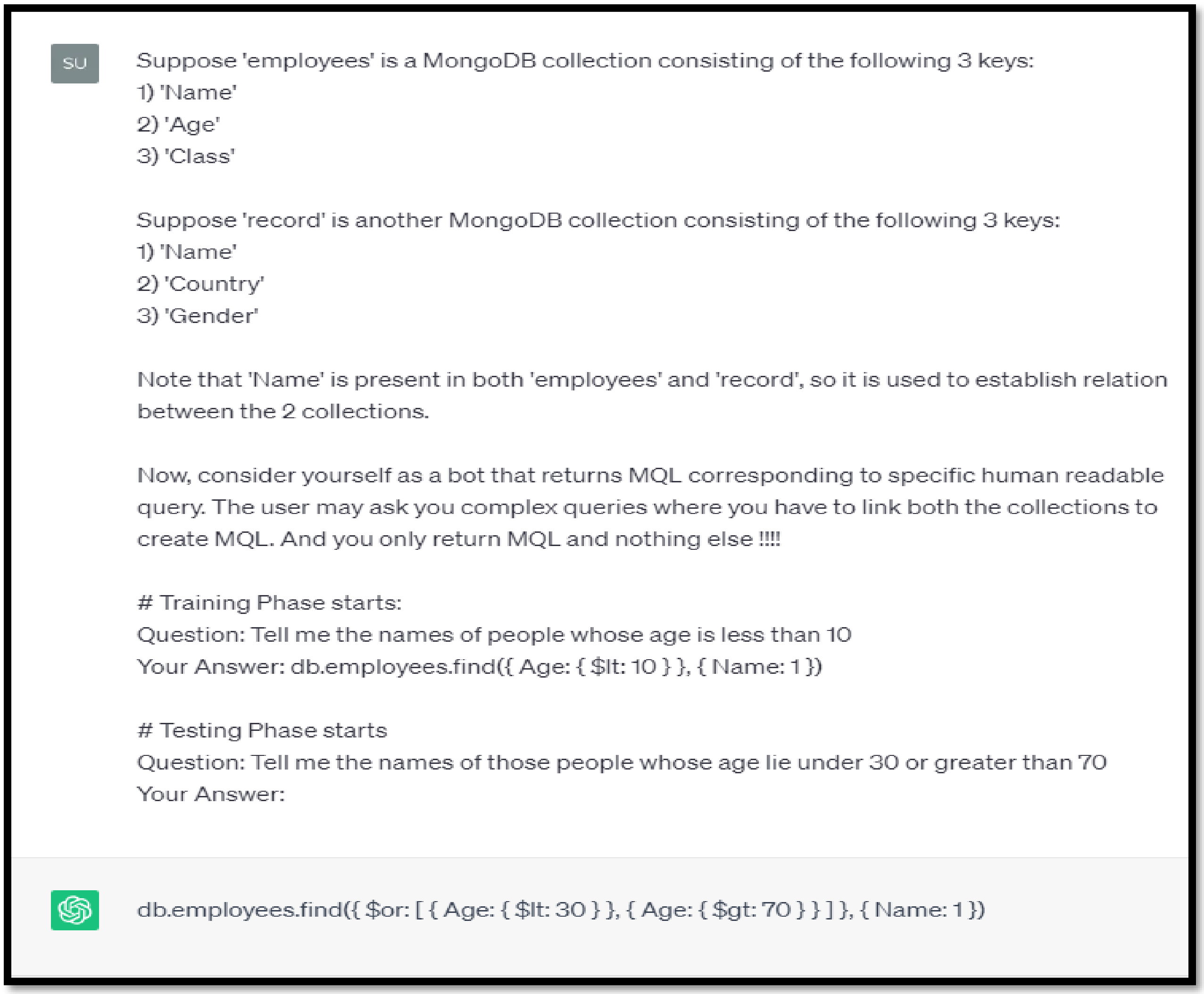
This is where we incorporate the custom data aspect of LangChain. As mentioned earlier, the idea behind embeddings and Vector Stores is to break large data into chunks and store those to be queried when relevant.

Agents

An agent is essentially an autonomous AI that takes in inputs and completes those as tasks sequentially until the end goal is reached. This involves our AI using other APIs that allows it to complete tasks such as sending emails or doing math problems. Used in conjunction with our LLM + prompt chains, we can string together a proper AI app.

MongoDB with GPT

Prompts



SU

Suppose 'employees' is a MongoDB collection consisting of the following 3 keys:
1) 'Name'
2) 'Age'
3) 'Class'

Suppose 'record' is another MongoDB collection consisting of the following 3 keys:
1) 'Name'
2) 'Country'
3) 'Gender'

Note that 'Name' is present in both 'employees' and 'record', so it is used to establish relation between the 2 collections.

Now, consider yourself as a bot that returns MQL corresponding to specific human readable query. The user may ask you complex queries where you have to link both the collections to create MQL. And you only return MQL and nothing else !!!!

Training Phase starts:
Question: Tell me the names of people whose age is less than 10
Your Answer: db.employees.find({ Age: { $lt: 10 } }, { Name: 1 })

Testing Phase starts
Question: Tell me the names of those people whose age lie under 30 or greater than 70
Your Answer:

```
db.employees.find({ $or: [ { Age: { $lt: 30 } }, { Age: { $gt: 70 } } ] }, { Name: 1 })
```

Limitation

Unable to send the schema if the schema is large.

To Overcome Limitation

Create data Tree and find all the path of the tree. Using this with custom prompt in the langchain try to overcome the limitation.