Week _1 GenAI using LangChain

Introduction

Foundation models, large data, big systems hardware, very expensive training, generalize (not task specific)
Eg: LLM = Foundation models

GenAI has only two parts:

- **User perspective** (use the foundation model to solve the problem)
  E.g : Prompt Engineering, Rag, AI-Agents, Vector Databases
- **Builder perspective** (you build the foundation model and deploy)
  E.g : RLHF, Pre-Training, Quantizing (helps to optimize the model and is used in different environments)

Builder Perspective:

- **Transformer Architecture**
- **Types of Tranformers** (Encorder(Bert),Decorder Only(GPT)) , Encoder and Decorder based (T5)
- **Pretraining** (Training Objectives, Tokenization Strategies, Training Strategies)
- **Optimization** (Training Optimization, Model Compression)
- **Fine-Tuning** (Task Specific Tuning (RLHF), Instruction Tuning (PEFT), Continual Training)
- **Evaluation**
- **Deployement**

User perspective

- **Building LLM Apps**
- **Using LLMs APIs**
- **Use Langchain**
- **Use Hugging Face**
- **Improve Response**
  Prompt Engineering
  RAG
  Fine Tunning
- **Agents**

Doing Tasks for yourself (eg: Ticket Booking)

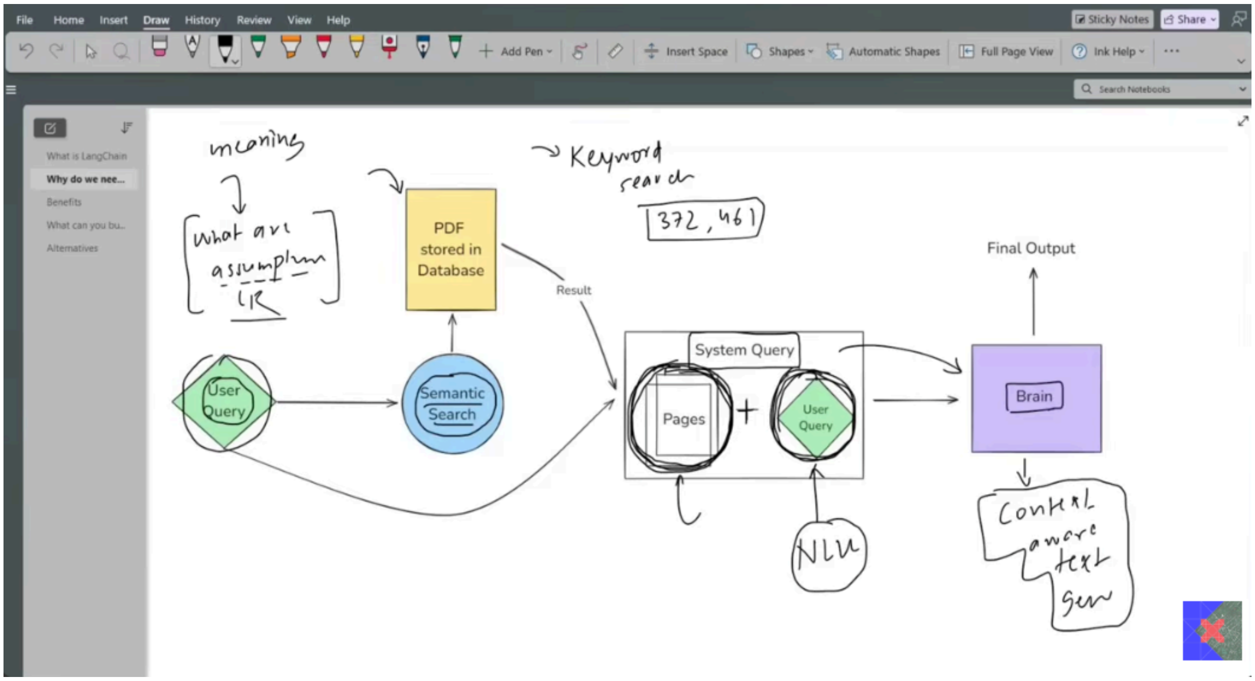
- **LLMOOPs**
- **Miscellaneous**

Langchain

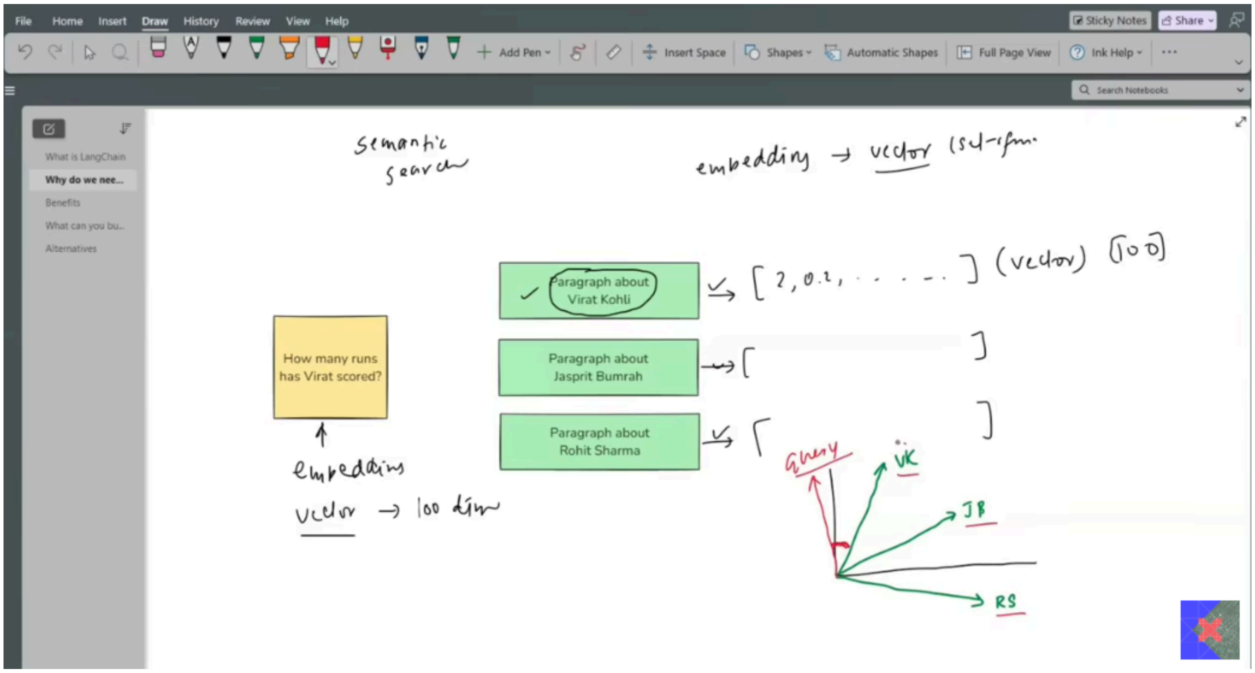
- It is an open-source framework that users can use for building LLM applications
- Chains (used to make complex pipelines)
- Integration is available for major tools
- Free
- Support all Majon GenAI use cases

Lecture 1: Introduction to LangChain

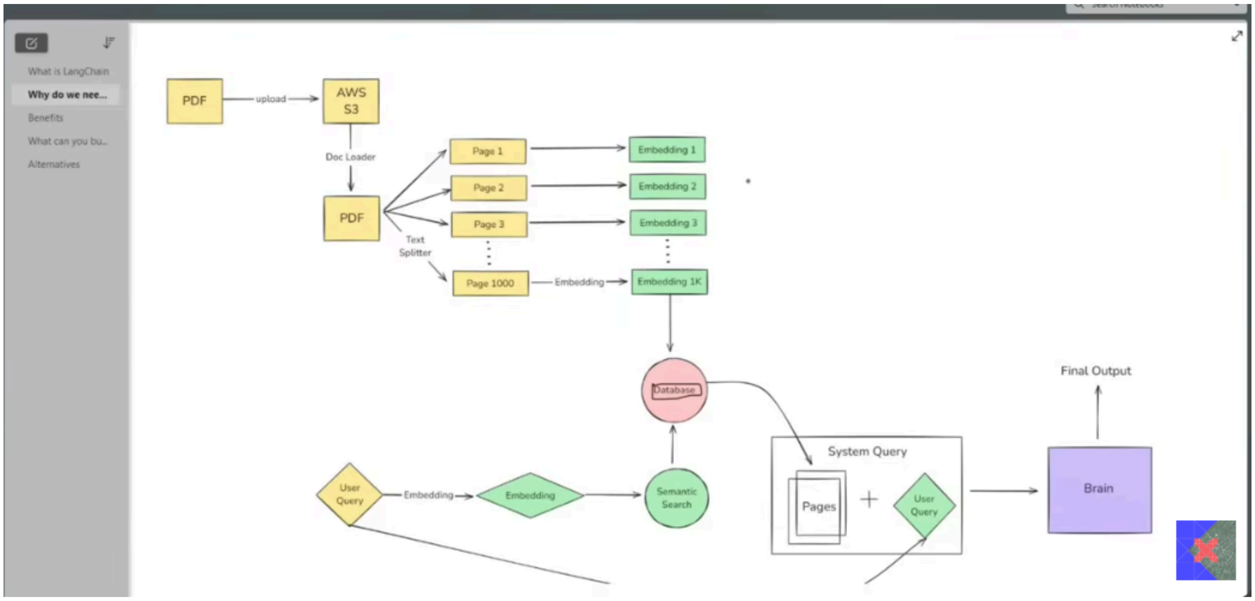
Old System Planned in 2014



Semantic Search



How Does the System Actually Work?



Challenges in implementing this system?

- Build Brain (a component that understands every query and generates relevant text)
  Cracked using LLM (Now it is not a big challenge)
- Deployment to server (manage cost + Computational Power + Engineering)
  Many companies solve this, like OpenAI (deployed to their server and offering their APIs)
- Storage

  Component of System

  AWS/GCP store document
  Text Splitter Model
  Embedding Model
  Embedding Database
  LLM

  Tasks of the System

  Document Loading
  Text Splitting
  Embedding
  Database management
  Retrivel
  Talk with the LLM model

  All tasks are executed using a pipeline

What does Langchain use in this system?

Langchain gives you built-in functionalities (Like you interact with components plugingly)

Benefits of Langchain

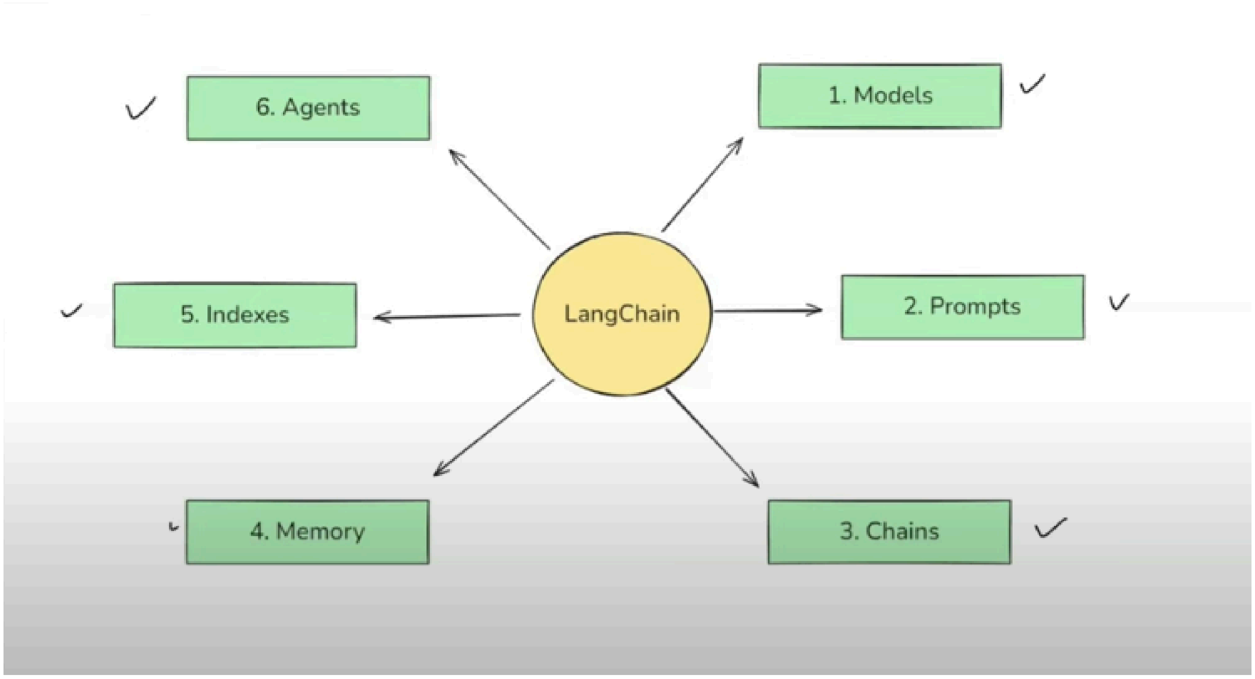
- **Concept of chains:**
  In a chain structure, we have different tasks and components, and execute a complex series of tasks
- **Model Agnostic Development:** (use any model), just focus on core business logic
- **Complete Ecosystem:** (Almost everything available, different embedding models, splitting models, or vector databases for integration)
- **Memory and State Handling:** In conversation memory

What can we build?

- **Conversational Chatbots** (eg: for company support)
- **AI knowledge Assistant** (Trained on specific data)
- **AI Agents** (Not Just Chat, It do Tasks Like Tickets booking)
- **Workflow Automation**
- **Summarization / Research helper (**to avoid giving private data to local chatbots**)**

Lecture 2: LangChain Components

Components of Langchain



Models

Models are the **core interfaces** through which you interact with AI models.

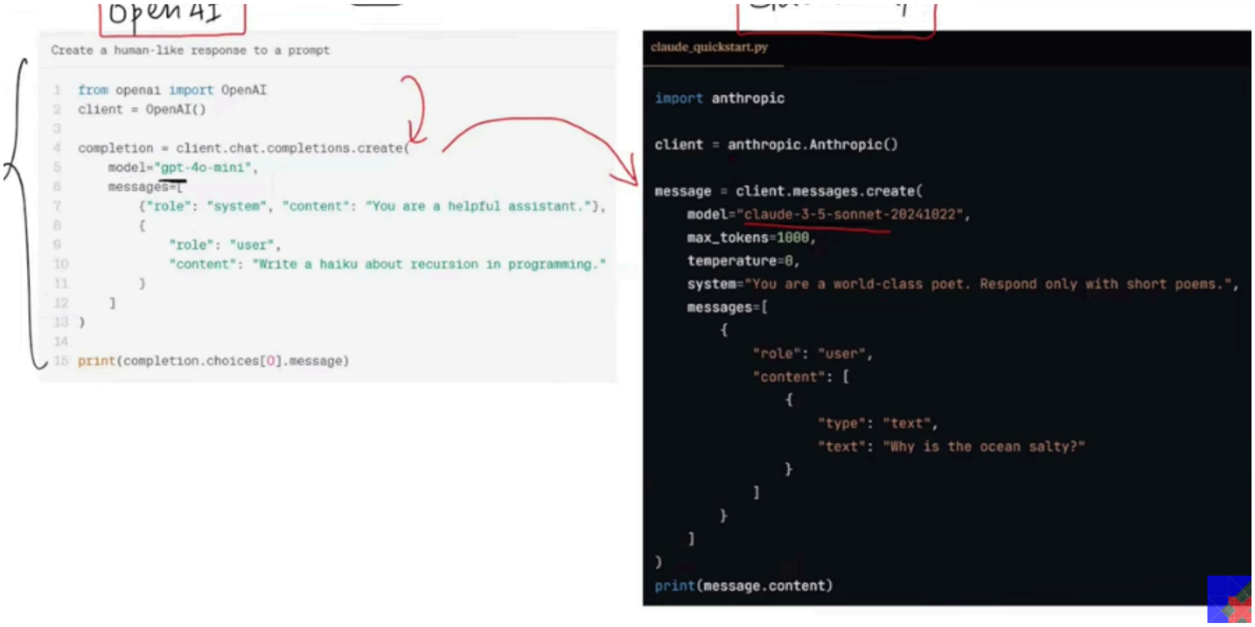
Past

In the past, NLP was used mostly for chatbots, but there are **major challenges.**

- **NLU**
- **Context-aware Text Generation**

But it is solved by LLM (but there is another problem LLM has, billions of parameters, we can't run it on a normal computer or a normal server), but it is solved by LLM APIs

Challenges in the Implementation of LLM APIs: Differences in communication with LLMs



Create a human-like response to a prompt

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {
            "role": "user",
            "content": "Write a haiku about recursion in programming."
        }
    ]
)

print(completion.choices[0].message)
```

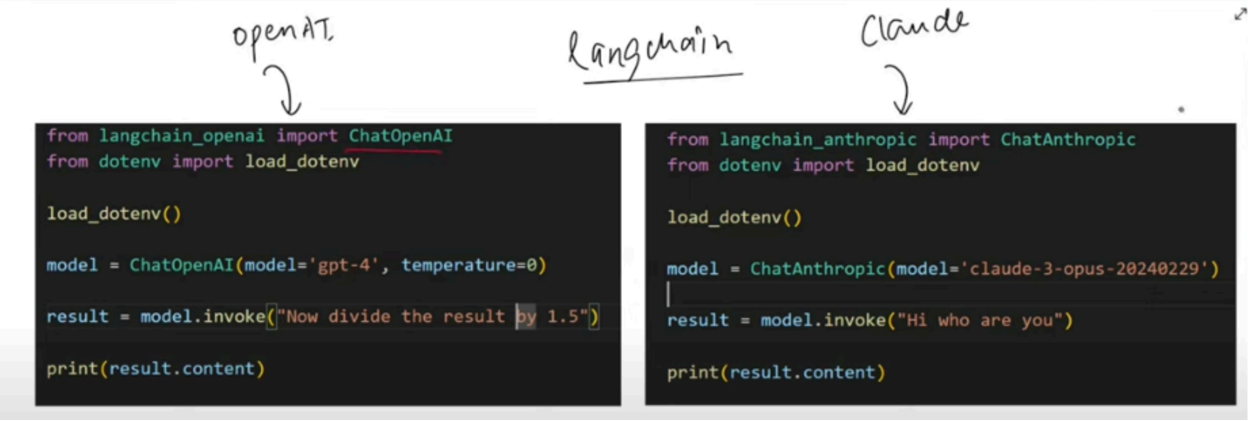
claude_quickstart.py

```
import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
    model="claude-3-5-sonnet-20241022",
    max_tokens=1000,
    temperature=0,
    system="You are a world-class poet. Respond only with short poems.",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "Why is the ocean salty?"
                }
            ]
        }
    ]
)
print(message.content)
```

Implementation of OpenAI and Anthropic Claude Models without using LangChain

How does Langchain solve the implementation of LLMs?

Langchain identifies that it solves this problem using a core model interface that helps in communication with different companies' models in a standardized way.



```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

model = ChatOpenAI(model='gpt-4', temperature=0)

result = model.invoke("Now divide the result by 1.5")

print(result.content)
```

```python
from langchain_anthropic import ChatAnthropic
from dotenv import load_dotenv

load_dotenv()

model = ChatAnthropic(model='claude-3-opus-20240229')

result = model.invoke("Hi who are you")

print(result.content)
```

Implementation of OpenAI and Anthropic Claude Model using langchain

Type of Models in Langchain

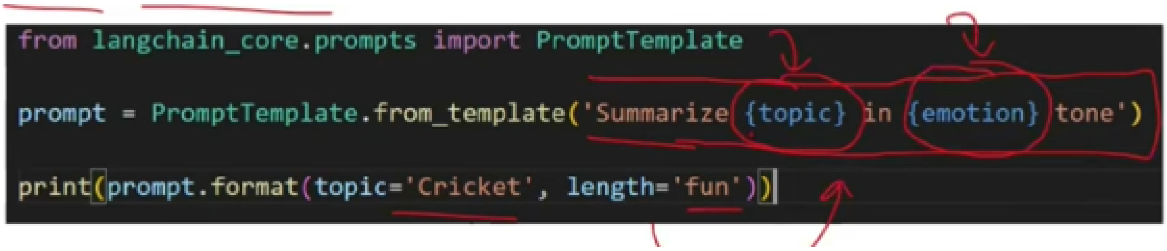
In Lagchain, we communicate with two types of models

- **Language Models:** (LLMs) Like we send text as input, and it returns text as output
  Chat Models
- **Embedding Models:** Models send text as input and give a vector as output, used for **semantic search**
  Embedding Models

Prompts

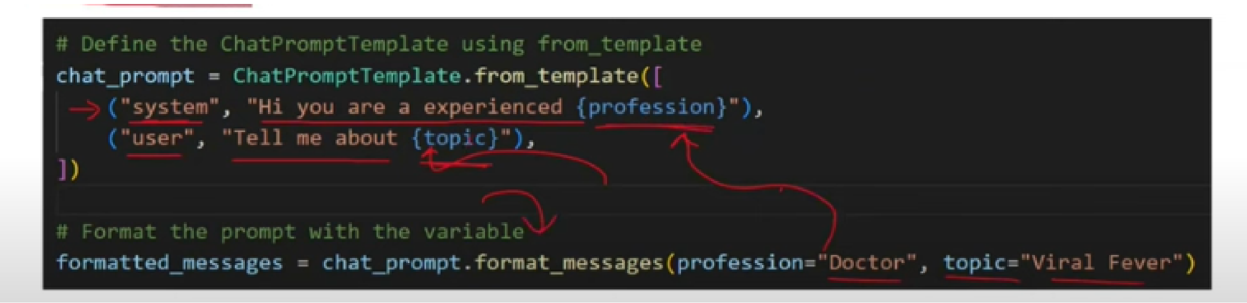
The input provided to LLM is the Prompt. The Output of LLM is very dependent on the prompt. Langchain designed a component, Prompt.

Dynamic and Reusable Prompts



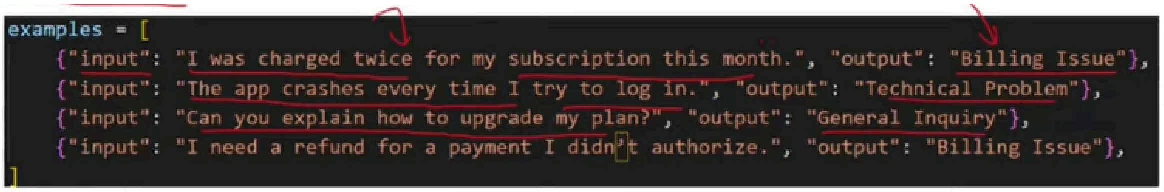
```python
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate.from_template('Summarize {topic} in {emotion} tone')

print(prompt.format(topic='Cricket', length='fun'))
```

Role-based prompts

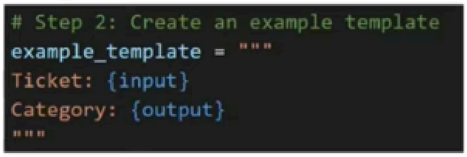


```
# Define the ChatPromptTemplate using from_template
chat_prompt = ChatPromptTemplate.from_template([
    ("system", "Hi you are a experienced {profession}"),
    ("user", "Tell me about {topic}"),
])

# Format the prompt with the variable
formatted_messages = chat_prompt.format_messages(profession="Doctor", topic="Viral Fever")
```

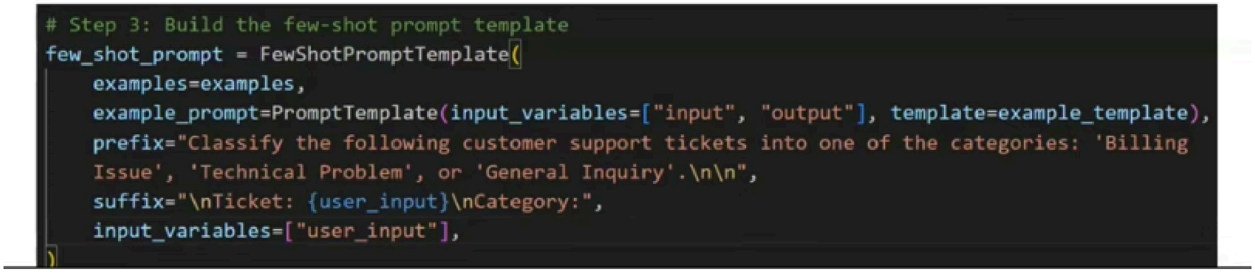
Few Short Prompts



```
examples = [
    {"input": "I was charged twice for my subscription this month.", "output": "Billing Issue"},
    {"input": "The app crashes every time I try to log in.", "output": "Technical Problem"},
    {"input": "Can you explain how to upgrade my plan?", "output": "General Inquiry"},
    {"input": "I need a refund for a payment I didn't authorize.", "output": "Billing Issue"},
]
```

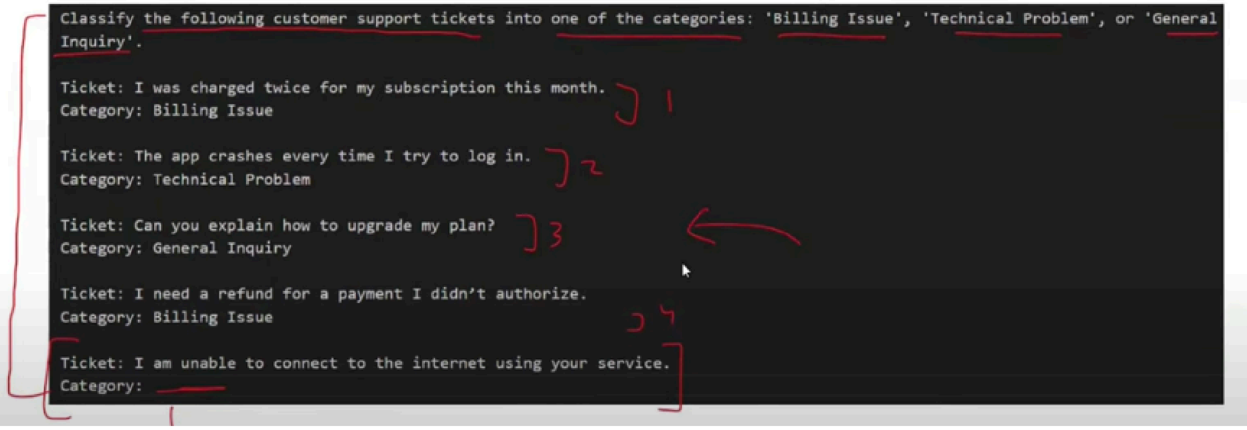


```
# Step 2: Create an example template
example_template = """
Ticket: {input}
Category: {output}
"""
```



```
# Step 3: Build the few-shot prompt template
few_shot_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=PromptTemplate(input_variables=["input", "output"], template=example_template),
    prefix="Classify the following customer support tickets into one of the categories: 'Billing
    Issue', 'Technical Problem', or 'General Inquiry'.\n\n",
    suffix="\nTicket: {user_input}\nCategory:",
    input_variables=["user_input"],
)
```

Prompt Designing



```
Classify the following customer support tickets into one of the categories: 'Billing Issue', 'Technical Problem', or 'General
Inquiry'.

Ticket: I was charged twice for my subscription this month.
Category: Billing Issue

Ticket: The app crashes every time I try to log in.
Category: Technical Problem

Ticket: Can you explain how to upgrade my plan?
Category: General Inquiry

Ticket: I need a refund for a payment I didn't authorize.
Category: Billing Issue

Ticket: I am unable to connect to the internet using your service.
Category:
```

Final Prompt hit to LLM

Chains

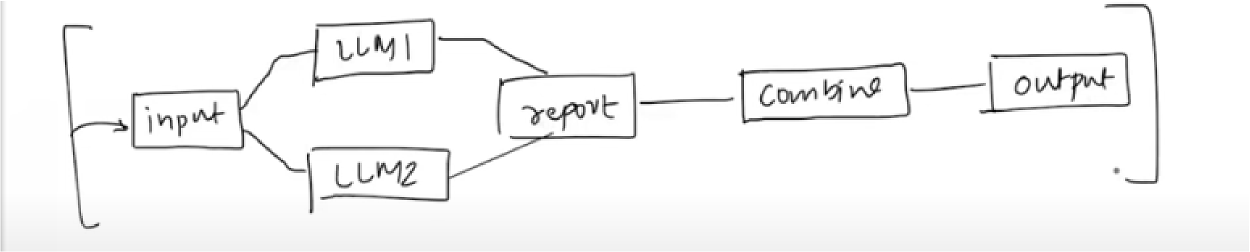
With the help of chains, we make pipelines in the LLM application.
E.g., the User gives input of 1000 words, and in the output, it gives a Hindi summary of less than 100 words.

Flow of Project
Input > LLM > Translate to Hindi > 2nd LLM > Hindi Summary Less then 100

- **In manually designing** without chaining every step and mapping every stage's input and output
- **Using a chain** for every step and executed automatically with the help of chain pipelines without writing extra code, and every stage component output is set as input for the next stage.

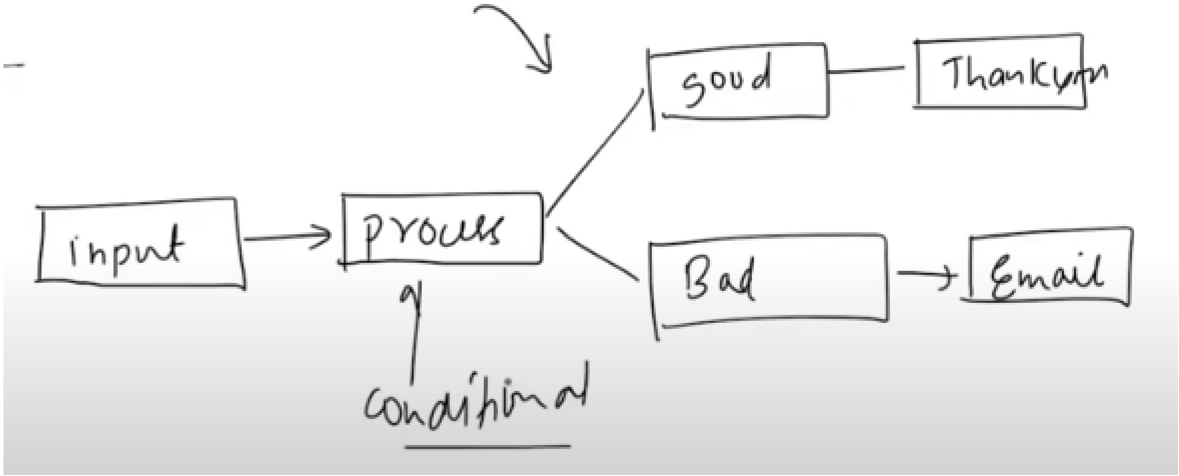
Parallel Chains

E.g., A system takes a keyword and generates a whole report like the incident of 9/11



Conditional Chains

E.g.: AI agent feedback processing



Indexes

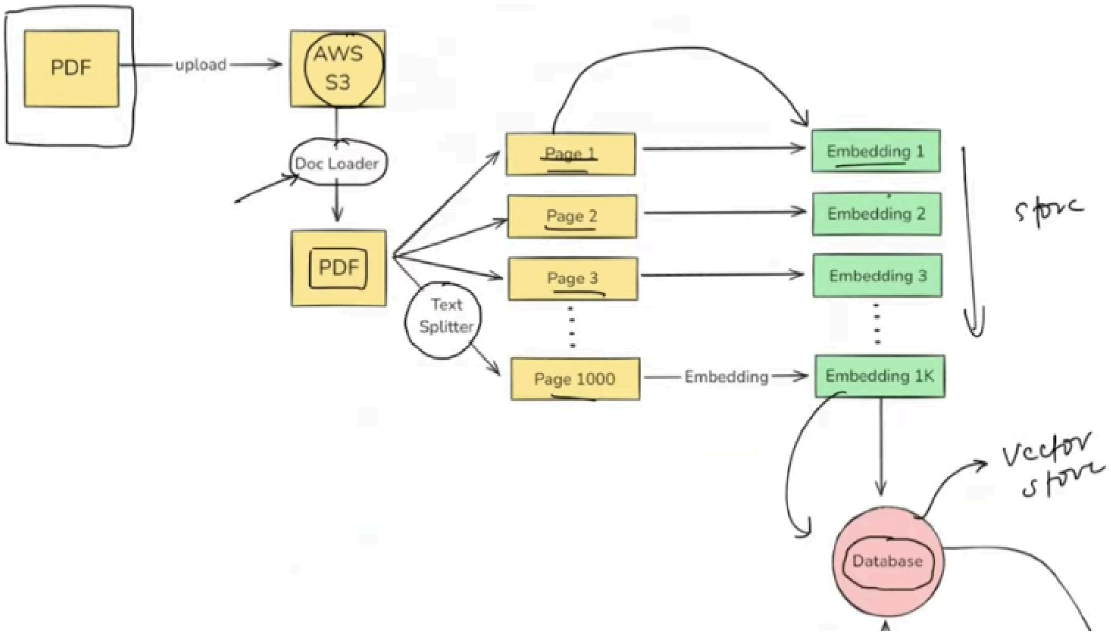
Indexes connect your application to external knowledge such as PDFs, databases, and websites.

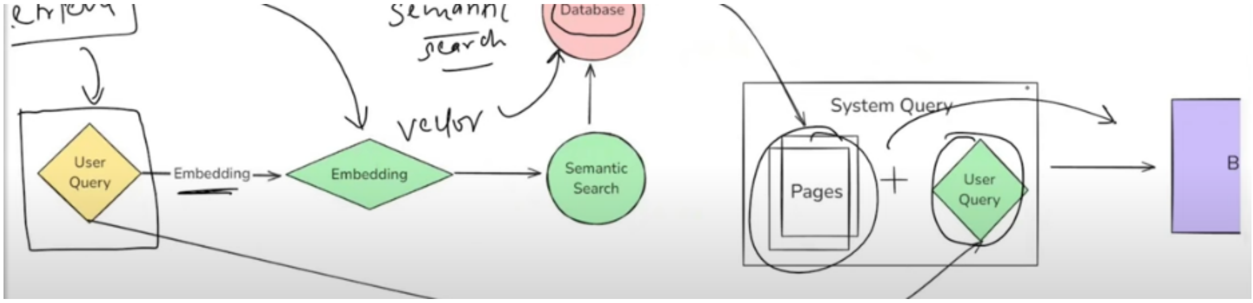
Contain

- Document Loader
- Text Splitter
- Vector Store
- Retrivers

ChatGPT was trained on the whole internet data. We query them about our XYZ company's privacy policy. Is GPT an answer to it? The answer is no.

To solve this problem, we give our company XYZ all private data and the rule book to LLM, then we query it, and it answers all our questions related to our company.

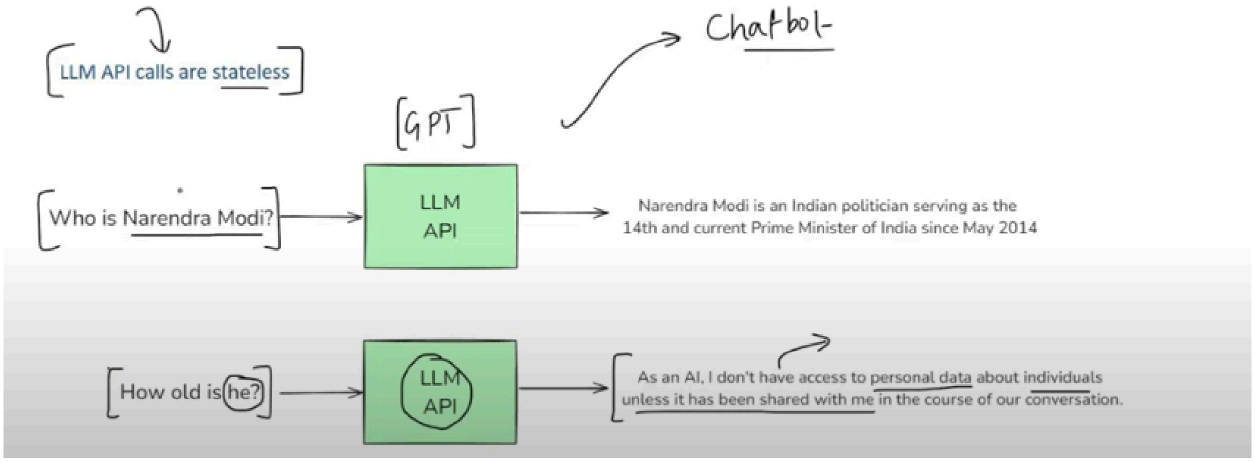




In short, Indexes is the way using which we build LLM applications which has access to an external knowledge source.

Memory

LLM api calls are **stateless** (which means they don't remember the previous api call). Solved by the Langchain memory component.



Narendra Modi is an Indian politician serving as the 14th and current Prime Minister of India since May 2014

As an AI, I don't have access to personal data about individuals unless it has been shared with me in the course of our conversation.

Types of memories frequently used in Langchain:

- **Conversation buffer memory :** (store transcript of last chats)
- **Conversation buffer window memory :** (store last n interactions of chat history)
- **Summarizer-based memory :** (periodically summarize old chat and store)
- **Custom memory :** (for advanced use causes eg: user preferences and key facts about them)

Agents

LLM (NLU+Text Generation) Like Chatbots
Agents have capabilities to do some tasks, we say a chatbot with some superpower, eg, booking tickets for you.

AI agents have

- Reasoning Capabilities
- Some Tools

Example agent

Our agent has this tool

- Calcuator
- Weather API

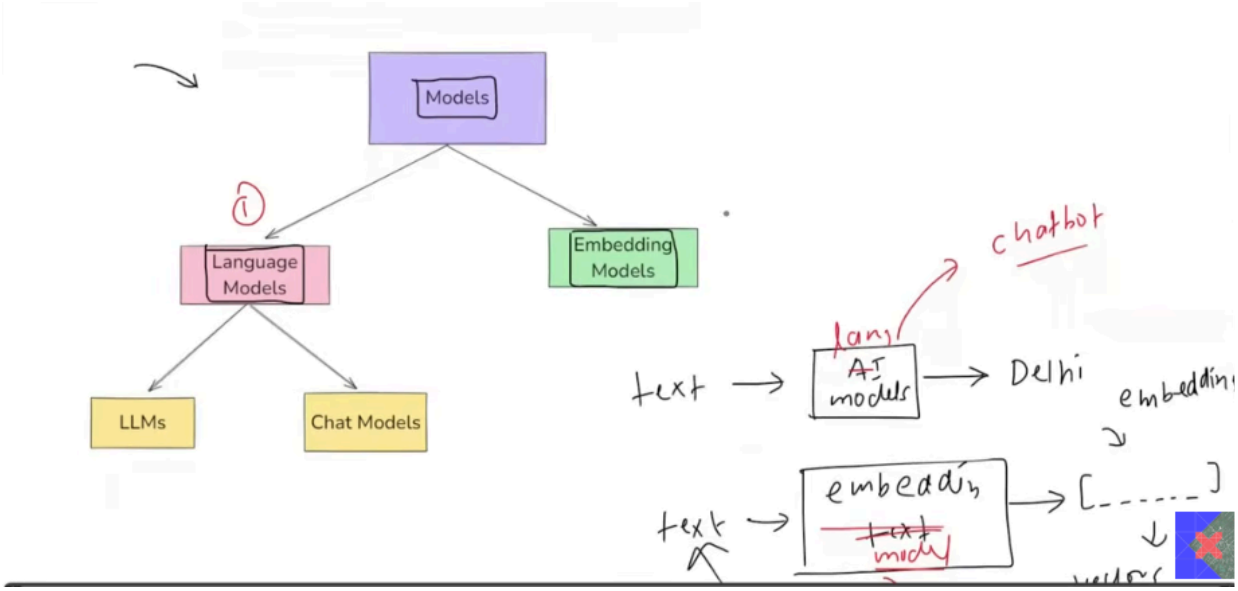
User Query: Can you multiply today's Lahore temperature by 3?

Working:

- **First,** break down the user query and understand what it actually does. Normally, agents have a thought of a reasoning tool.
- **The second** user hit the weather api and got today's Lahore temperature, like 41.
- **Third** multiply temperature by 3, so it uses the calculator and multiplies temperature 41*3 and returns 123 as the final output.

Lecture 3: Models

The Model Component in Langchain is a crucial part of the framework designed to facilitate interactions with language models and embedding models. Models are the **core interfaces** through which you interact with AI models.



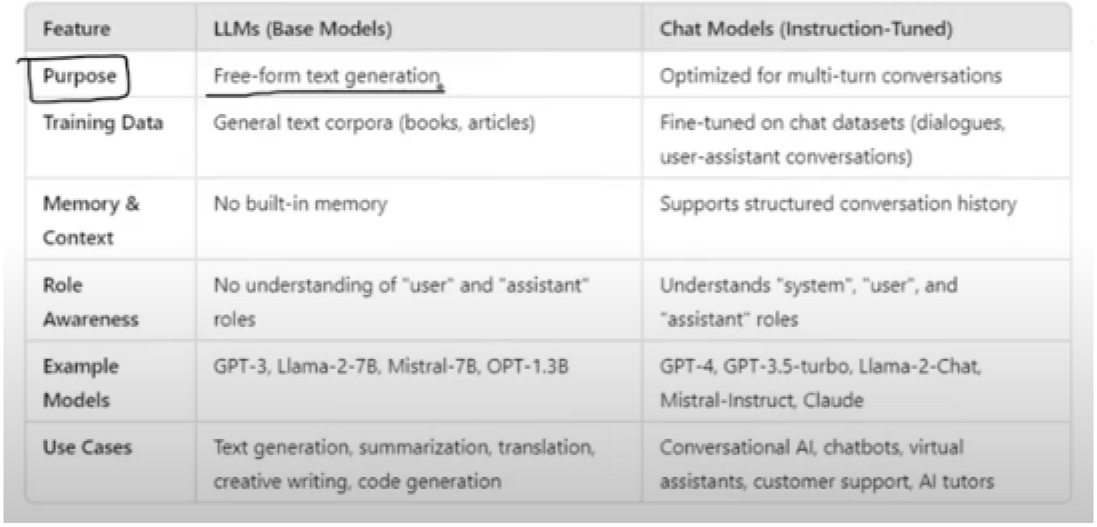
Types of Models We Use

Language Models

The models in which we sent a text as input to the model process, and gave a text as output.

Types of Language Models

- **LLM:** A general-purpose model used for any type of NLP application, used for text generation, text summarization, code generation, and question answering.
- **Chat-Model:** Chat models are language models used for conversational tasks; they take a sequence of messages as input and return chat messages as output. The major difference between LLMS and chat models is that chat models support conversation history.



Feature	LLMs (Base Models)	Chat Models (Instruction-Tuned)
Purpose	Free-form text generation	Optimized for multi-turn conversations
Training Data	General text corpora (books, articles)	Fine-tuned on chat datasets (dialogues, user-assistant conversations)
Memory & Context	No built-in memory	Supports structured conversation history
Role Awareness	No understanding of "user" and "assistant" roles	Understands "system", "user", and "assistant" roles
Example Models	GPT-3, Llama-2-7B, Mistral-7B, OPT-1.3B	GPT-4, GPT-3.5-turbo, Llama-2-Chat, Mistral-Instruct, Claude
Use Cases	Text generation, summarization, translation, creative writing, code generation	Conversational AI, chatbots, virtual assistants, customer support, AI tutors

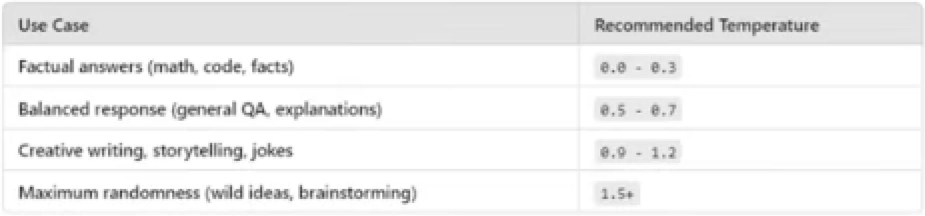
In the models component of LangChain, the **Invoke function** is used to send prompts to models.

For checking the OpenAI available models: https://platform.openai.com/docs/models

Parameters of Models:

Temperature is the parameter of the model that controls the randomness of a language model, which affects how creative and deterministic the response is

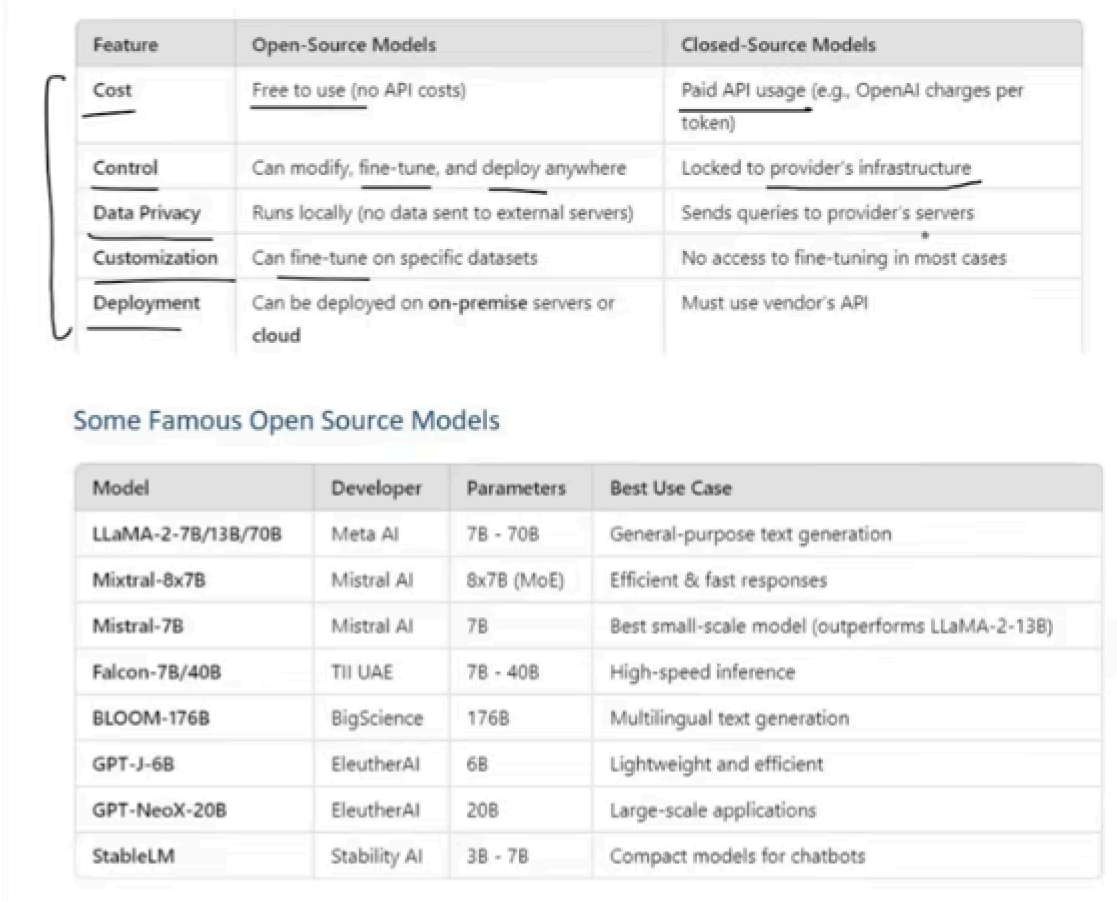
- **Lower Value** (0.0 to 0.3) is more deterministic and predictable
- **Higher Value** (0.7 to 1.5), more random and creative



Use Case	Recommended Temperature
Factual answers (math, code, facts)	`0.0 - 0.3`
Balanced response (general QA, explanations)	`0.5 - 0.7`
Creative writing, storytelling, jokes	`0.9 - 1.2`
Maximum randomness (wild ideas, brainstorming)	`1.5+`

Max_completion_tokens is used for specifying the response token, or we say words.

Open Source Models



Feature	Open-Source Models	Closed-Source Models
Cost	Free to use (no API costs)	Paid API usage (e.g., OpenAI charges per token)
Control	Can modify, fine-tune, and deploy anywhere	Locked to provider's infrastructure
Data Privacy	Runs locally (no data sent to external servers)	Sends queries to provider's servers
Customization	Can fine-tune on specific datasets	No access to fine-tuning in most cases
Deployment	Can be deployed on **on-premise** servers or **cloud**	Must use vendor's API

Some Famous Open Source Models

Model	Developer	Parameters	Best Use Case
LLaMA-2-7B/13B/70B	Meta AI	7B - 70B	General-purpose text generation
Mixtral-8x7B	Mistral AI	8x7B (MoE)	Efficient & fast responses
Mistral-7B	Mistral AI	7B	Best small-scale model (outperforms LLaMA-2-13B)
Falcon-7B/40B	TII UAE	7B - 40B	High-speed inference
BLOOM-176B	BigScience	176B	Multilingual text generation
GPT-J-6B	EleutherAI	6B	Lightweight and efficient
GPT-NeoX-20B	EleutherAI	20B	Large-scale applications
StableLM	Stability AI	3B - 7B	Compact models for chatbots

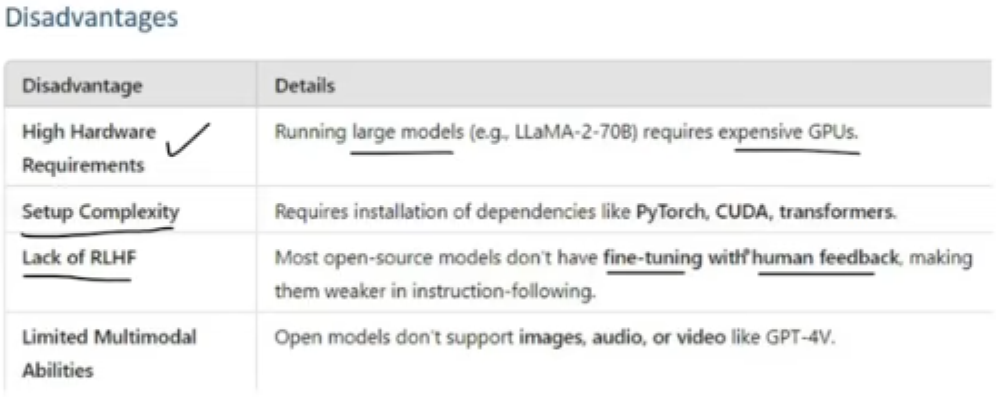
Where do we find open-source models?

Hugging Face

Way to Use Open Source Models

- Using Inference Api (HF provides a free api like OpenAI AI, also with a free tier)
- Run Locally

Disadvantages of open-source models



Disadvantages

Disadvantage	Details
High Hardware Requirements ✓	Running large models (e.g., LLaMA-2-70B) requires expensive GPUs.
Setup Complexity	Requires installation of dependencies like **PyTorch, CUDA, transformers**.
Lack of RLHF	Most open-source models don't have **fine-tuning with human feedback**, making them weaker in instruction-following.
Limited Multimodal Abilities	Open models don't support **images, audio, or video** like GPT-4V.

How to Use Hugging Face Inference Api

- Redirect to "https://huggingface.co/"
- Go to your profile > Access Token
- Then click on "Read" if you just use the model, and don't want to fine-tune it.
- Then copy the access token key and use it in your project