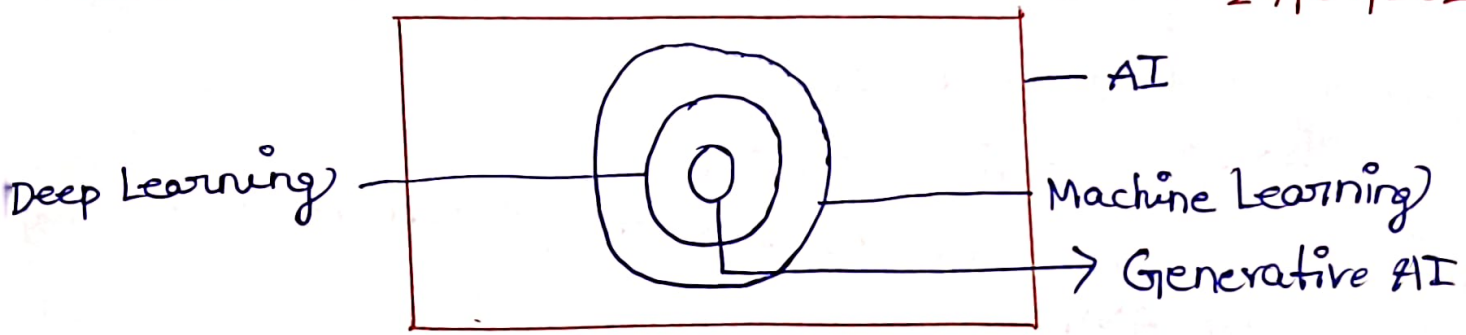
Generative AI

27/01/2025



AI → Build applications that can perform its own task with human intervention.

Ex: Netflix → Recommendation System
Self driving cars

AI Engineer → Integrating the model involves fine tuning etc....
→ Creating AI product.

ML → Machine Learning is the subset of AI.
→ TO provide Statstools to perform :-
* Statistical Analysis, Visualization, prediction and forecasting.
→ Data injection → Data transformation
→ Feature Engineering
→ understanding the data.

DL → Built to mimic human brain
→ Multi Layered Neural Network.

DL → ANN
→ CNN and object detection
→ RNN and its variants

CNN → Computer vision

RNN → Text Related usecases
↳ Time series usecases.

ANN → We can train ML problem statement with help of ANN.

{transformers and Bert} Very Advanced.

⇓

{Generative AI}

Generative AI → Subset of Deep Learning

Models :- → Discriminative AI (or) Model
→ Generative AI or Model

Discriminative Model

Task → classification, prediction

Dataset → Trained on Labelled Dataset

Generative AI

Task → Generates new data trained on huge dataset

In Gen. AI two types of models:-

① LLM → Large-Language Model ⇒ Text Data

② LIM → Large - Image Model ⇒ Images

LLM:- Open AI, Meta, Google, Anthropic

↓ GPT 4 ↓ LLama2 ↓ Gemini ↓ Claudy 3

↓ Gemma

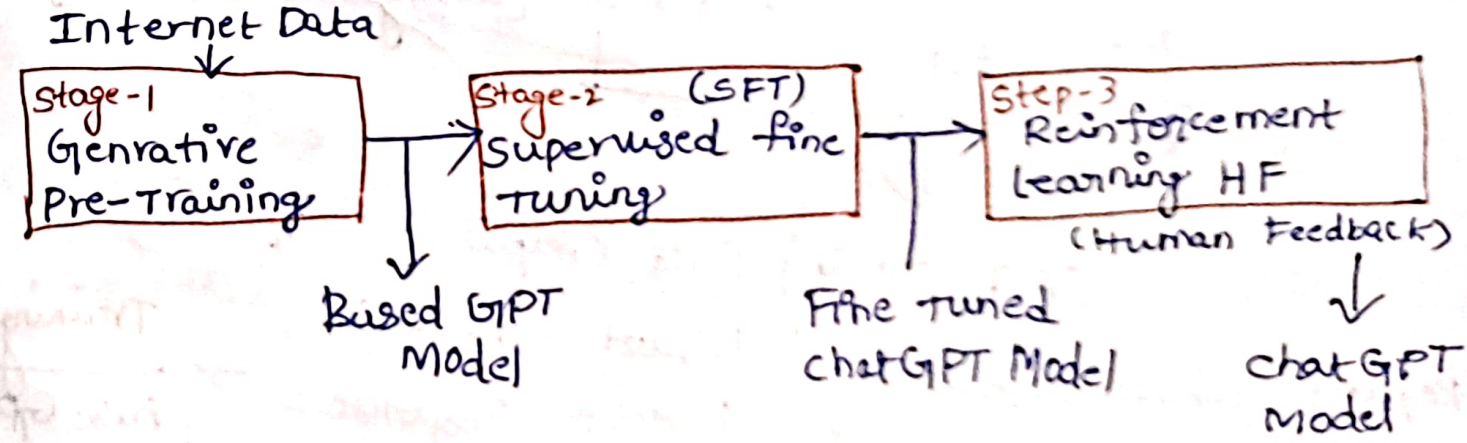
→ Foundation Models } Trained on
→ pretrained Models } Huge Data

Langchain:- It is a frame work

RAG Application:- Retrieval Argumentation Query xxx

↳ Retrivel Argumented Generation. ⇒ develop chatbot

How chat GPT is Trained :-



① Stage-1 ⇒ [Generative Pre Training]

Internet Huge Input Text Data → [Transformers] → Base GPT Model

↓

What we want ?

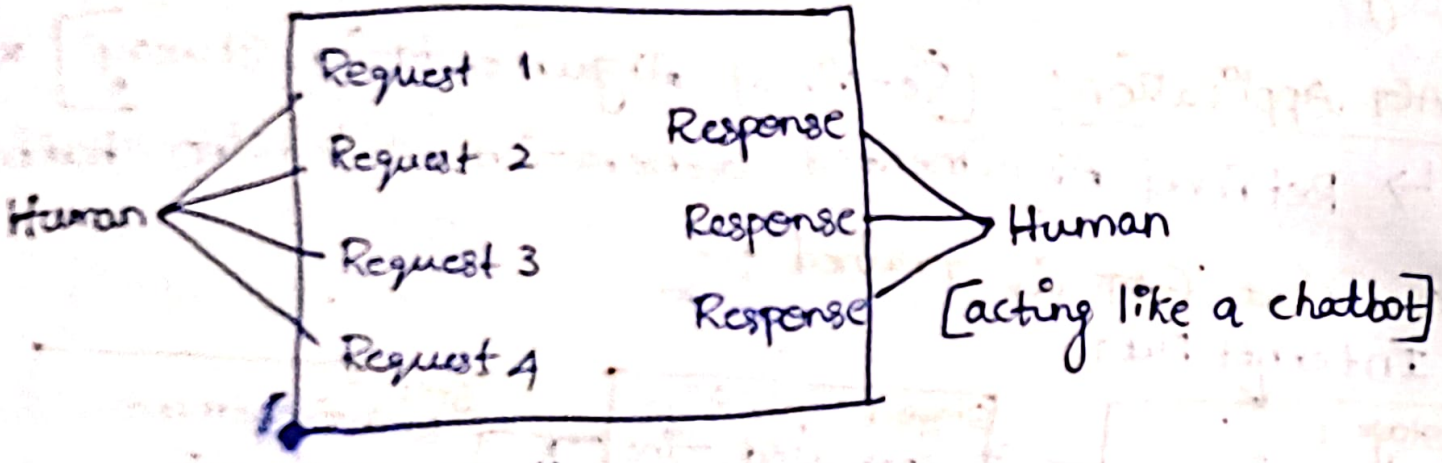
[conversation] ⇒ chatbot

Tasks:-

1. Language Translation
2. Text Summarization
3. Text completion
4. Sentiment Analysis

② Stage-2 [Supervised Fine Tuning] (SFT)

[Real Conversation]



⇓

SFT Training in Data Corpus

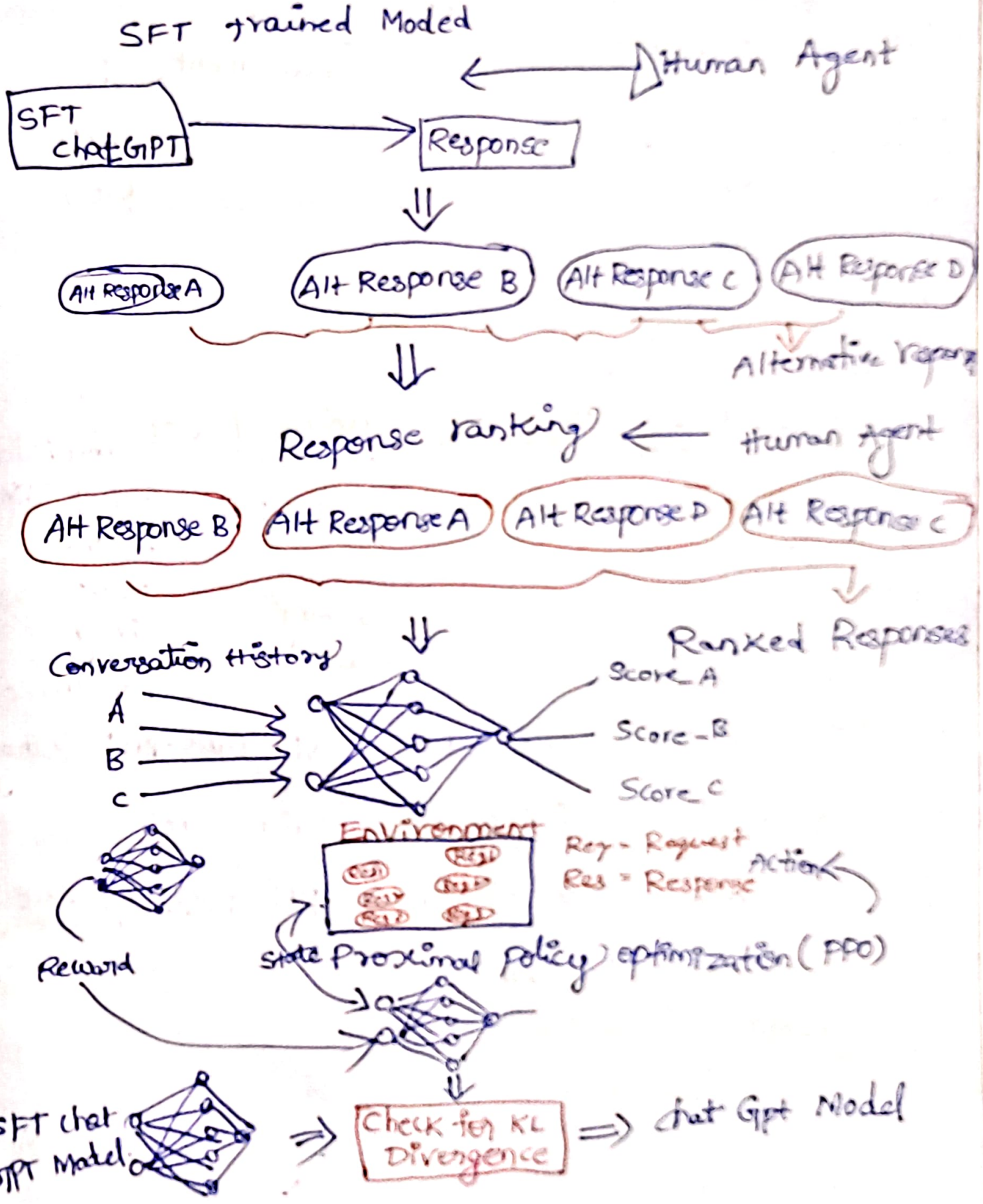
Request = conversation history

Response = Best Ideal Response

Request	Response 1
Request	Response 2
Request	Response 3

⇒ Training → Base GPT Model → Optimizes [SGD] → SFT Chat GPT Model

③ Stage - 3 = [Reinforcement Learning through Human Feedback (RLHF)]



Evalution of LLM (Large Language Models)

1967 Eliza ⇒ 1970 SHRDLU ⇒ 1980 XCALIBU ⇒ 1988 RNN (Advancement) ⇒ 1997 LSTM

⇓

2014 GRU ⇒ 2017 Transformers ⇒ 2018 Bert, GPT ⇒ 2019 GPT-2, RoBERTa, XLNet

⇓

2020 GPT-3 (open AI) ⇒ 2021 GPT-3.5 ⇒ 2022 PaLM, Instruct GPT, Chat GPT (Fine tuning Models) ⇒ 2023 LLama, Falcon, GPT-4, LIMA, PaLM 2, BARD, Dolly 2, Guanaco

2024 → open AI → Multimodal → GPT 4.0 ⟷ NLP → Images

2024 → Google gemini Pro 1.5 } paid Models

2024 → Llama 3, Anthropic Claude opus, Hugging face

⇒ Google Gemma 2

Langchain Ecosystem :-

→ why Langchain

→ LangChain Ecosystem ?

⇒ Langchain is the most common framework that is specifically used to build Generative AI application

Langchain :- open source

→ Frame work to develop Gen AI app.

Gen AI App → LLMOPS [Debugging, Evalution, Monitoring]

Langsmith {observability} → Debugging → playground → Evolution
→ Annotation Monitoring

LangServe {Deployments} → chains as Rest API's

Templates → Reference Applications

Langchain → chains, Agents, Retrieval Strategis {Cognitive Architecture}

Langchain-community → {Integration components}
Model I/O
→ model
→ prompt
→ Example selector
→ output parser

Retrieval
→ Retriever
→ Document loader
→ Vector Store
→ Text Splitter
→ Embedding model

Agent Tooling
→ Tool
→ Toolkit

langchain core → {protocol}
LCEL → Langchain Expression Language

→ parallelization → Fallbacks → Tracing → Batching

→ Streaming → Async → composition.

→ open 'cmd'

→ 'cd' 'folder path'

→ d:

→ code . (code space dot)

.env →OPENAI_API_KEY = " "

→ LANGCHAIN_API_KEY = " "

→ LANGCHAIN_PROJECT = " "

Components of Langchain:-

① RAG → Retrieval Argumented Generation

Loading the Dataset → Diagram / PDF / Image / URL / JSON

[Data Ingestion]

⇓

Text or document

Splitting the data into smaller chunks

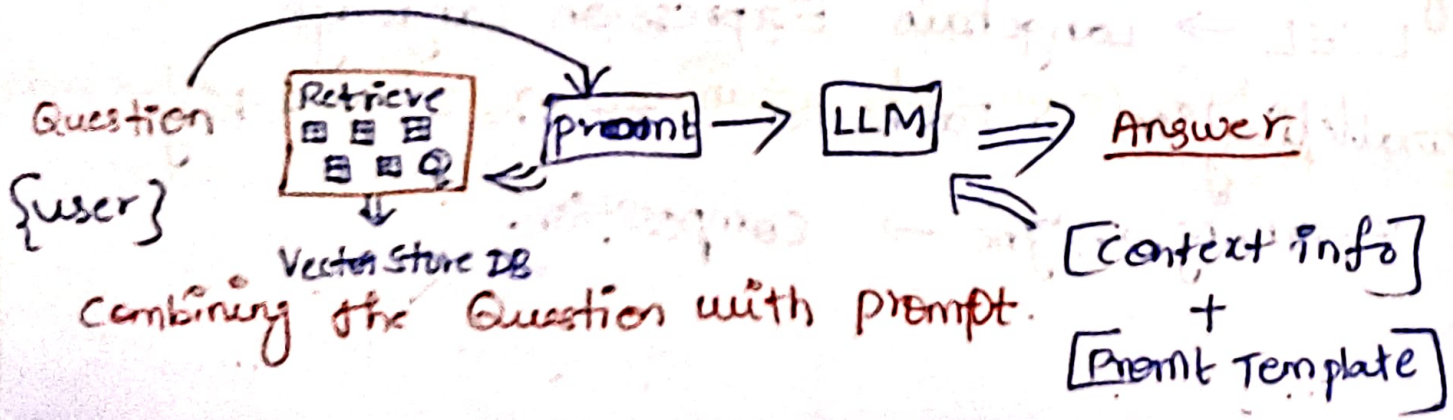
⇓

Embedding the text into vectors

⇓

Storing the data into the Vector Store DB

Types:- FAISS / CMROMA DB / ASTRAD DB



Combining the Question with prompt.

→ Retrieval chain

⇓

Interface (responsible for Querying the Vectorstore DB)

⇓

Vector Store DB

Data - Ingestion :-

```
from langchain_community.document_loaders import TextLoader
loader = TextLoader("speech.txt")
text_documents = loader.load()
```

{Requirements.txt}
→ langchain
→ python-dotenv
→ ipykernel
→ langchain_community
→ pypdf
→ bs4
Beautiful Soup

```
from langchain_community.document_loaders
                    import PyPDFLoader
loader = PyPDFLoader('attention.PDF')
PDF_Documents = loader.load().
```

```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoade('URL')
WebLoader = Loader.load()
```

BS4 => It's a python library used for web Scraping and parsing HTML or XML document

```
from langchain_community.document_loaders import WebBaseLoader
import BS4
loader = WebBaseLoader(Web_Paths=('URL'),),
        bs_kwargs = dict(parse_only = bs4.SoupStrainer
        (class='post-title', 'post-content', 'post-header')))
```

WebBaseLoader = loader.load

Kwargs => Key word Aguments

BS4 Keyword arguments → parse_only
filter => (SoupStrainer)

Loading Data ingestion:-

→ Text
→ PDF
→ WebURL
→ Arxiv
→ Wikipidia.

```
from langchain_community.document_loaders import
                                        WikipediaLoader
wiki_Loader = WikipediaLoader(query="Generative AI",
                              load_max_docs=2).load()
len(wiki_loader)
```

Text splitter Techniques:-

① Recursive Text Splitter
② character Text Splitter
③ HTML Header Text splitter
④ Recursive Json splitter.

} Types of text splitter techniques.

EMBEDDING:-

① Open AI — paid
② OLLAMA — open source
③ Hugging face — open source

} Embedding Techniques

Embedding :—

Converting the text into vectors

⇒
```
import os
from dotenv import load_dotenv
load_dotenv( ) # load all environment variables
```

⇒ os.environ["OPENAI_API_KEY"]
= os.getenv("OPENAI_API_KEY")

⇒ from langchain_openai import OpenAIEmbeddings
embeddings = ~~OPENAI~~ OpenAIEmbeddings(model="text-embedding-3-large")
embeddings.

⇒ text = "This is a tutorial on OPENAI Embedding"
query_result = embeddings.embed_query(text)
query_result ⇒ Convert Text into vectors

⇒ len(query_result) ⇒ 3072

Embeddings_1024 = OpenAIEmbeddings(model="text_embedding-3-large", dimensions=1024)

⇒ text = "This is a tutorial on OPENAI Embedding"
query_result = Embeddings_1024.embed_query(text)

dimensions ⇒ Specifying the size of the Embedding vectors.

Lanchain_community. Vector stores

⇒ FAISS ⇒ Chroma ⇒ Pinecone ⇒ Weaviate

Ollama :- open source llm models [plotform]

→ LLama3, phi3, Mistral, Gemma

open source model ⇒ Langchain

In open source model Embedding techniques ⇒ ollama

Vector store DB :-

⇒ FAISS ⇒ Chroma DB. ⇒ Astra DB

FAISS :-

```
from langchain_community.document_loaders import TextLoader
from langchain_text_splitters import CharacterTestSplitter
from langchain_community.embeddings import OllamaEmbeddings
from langchain_community.vectorstores import FAISS
```

⇒ `Loader = TextLoader('speech.txt')`

`documents = loader.load()`

`text_splitter = CharacterTestSplitter(chunk_size=200, chunk_overlap=30)`

`docs = text_splitter.split_documents(documents)`

⇒ `Embeddings = OllamaEmbeddings()`

`db = FAISS.from_documents(docs, embeddings)`

`db`

Querying in FAISS :-

```
query = " "
docs = db.similarity_search(query)
docs[0].page_content
```

Retriever :-

```
retriever = db.as_retriever( )
docs = retriever.invoke(query)
docs[0].page_content
```

For this we get same data.

① Query

② Retriever

similarity search.

Invoke

similarity_search_with_score

```
docs_and_score = db.similarity_search_with_score(query)
docs_and_score.
```

Embedding vector :-

```
Embedding_vector = Embeddings.embed_query(query)
embedding_vector.
```

⇒ Components of Langchain :-

```
import os
from dotenv import load_dotenv
load_dotenv( )
```

⇒ Langsmith Tracking :-

```
os.environ["LANGCHAIN_API_KEY"]
        = os.getenv("LANGCHAIN_API_KEY")
os.environ["LANGCHAIN_TRACKING_V2"] = "true"
os.environ["LANGCHAIN_PROJECT"] = os.getenv("LANGCHAIN
                                    _PROJECT)
```

⇒ from langchain_community.llms import Ollama

```
llm = Ollama(model="llama2:7b")
print(llm)
```

⇒ response = llm.invoke("Explain Langchain in simple terms.")

⇒ response

Chatprompt Template :-

```
from langchain.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_message(
    [
        ("sys", " "),
        ("user", "{ }")
    ]
)
prompt
```

Chain :-

```
Chain = prompt | llm
response = chain.invoke({"input": "you can tell me about langsmith ?"})
print(response)
```

Simple App with Ollama - llama → Gen AI

```
import os
from dotenv import load_dotenv
load_dotenv
```

```
from langchain_community.llms import ollama
import streamlit as st
from langchain_core.prompts import ChatPromtTemplate
from langchain_core.input_parsers import StroutputParser

# Langsmith Tracking
# prompt template
# streamlit frame work
# Ollama Llama2 Model
```

⊕ LCEL => Langchain Expression Language

open source models => Llama3, Gemma2, mistral } platform Groq

```
from langchain_groq import ChatGroq
model = ChatGroq(model="gemma2-9b-it", groq_api_key
                 ="groq_api key")
from langchain_core.messages import HumanMessage, System
                                                  Message
messages = [
    SystemMessage(content="    "),
    HumanMessage(content="    ?")
]
result = model.invoke(messages)
result.
```

```
From langchain_core.output_parsers import StrOutputParser
parser = StrOutputParser()
parser.invoke(result)
```

#langchain Server:- (serve.py)

import ① FASTAPI
② chatprompt template
③ str Output parser
④ ~~fastapi~~ chat_Groq
⑤ os
⑥ dotenv → load_dotenv
⑦ add_routes (langserve)

POSTMAN → To check translation in langchain server.

Chatbot → ipynb:-

⇒ Building Chatbot with message history using Langchain

```
import os
from dotenv import load_dotenv

load_dotenv()

groq_api_key = os.getenv("GROQ_API_KEY")
groq_api_key
```

⇒ from langchain_groq import ChatGroq
model = ChatGroq(model="Gemma2-9b-it", groq_api_key = Groq)

⇒ from langchain_core.messages import HumanMessage
model.invoke([HumanMessage(content="")])

⇒ from langchain_core.messages import AIMessage
model.invoke(
[
HumanMessage(content="")
SystemMessage(content="")
HumanMessage(content="")
]
)

Message History :- (from langchain community)

from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory

Prompt chain :-

from langchain_core.prompts import ChatPromptTemplate, MessagePlaceholder
prompt = ChatPromptTemplate.from_messages(
[
("System", "")
MessagePlaceholder(variable_name="messages")
]
prompt | model

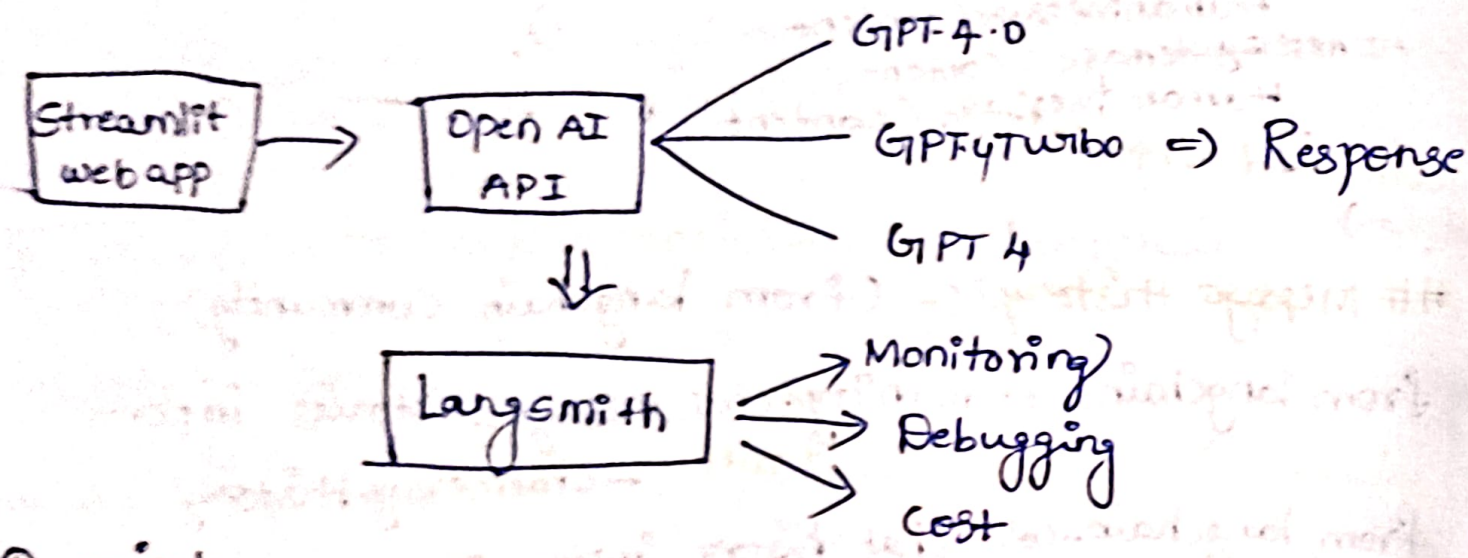
Trim_message :-

Parameters → max_tokens, strategy, token_counter, include_system, allow_partial, start_on

Q&A Chatbot

① Open AI LLM Models :- GPT-4 o , GPT- 4 Turbo

② OLLAMA → open source → LLAMA3, GEMMA2, Mistral



① project

② Environment Variable

③ Required libs

④ Streamlit web app

⑤ Deployment

Groq :-

Groq is a company known for developing ultra-fast AI accelerators and processors.

→ their technology is designed to run large language models.

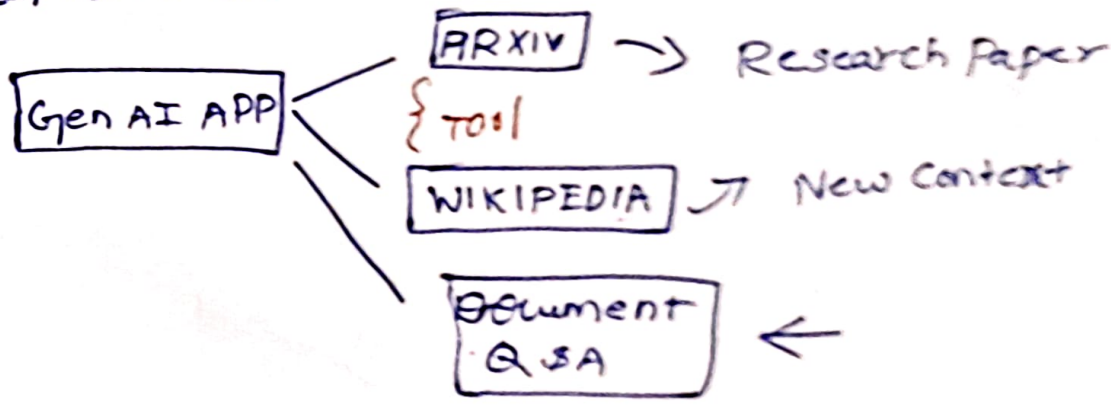
Rag :- [Retrieval-Agumented-Generation]

→ It is a smart AI technique that helps chatbots and AI Models give better, more accurate answers.

Tools and Agents :-

→ Tools are the interfaces that an agent, chain or llm can use to interact with the world.

Eg :-



Pine Cone :-

1. creating pinecone Account
2. Sign in
3. Creating the project name
4. copy the API Key
5. Creating the Index

configuration — Dimensions / Metric

Cosine
Euclidean
dot product

6. cloud provider
7. Region (aws - Virginia (open source))

Azure Machine Learning :-

25/02/2025

Azure ML is a cloud based platform for

→ Building
→ Training
→ Deploying
→ Managing Machine Learning Models.

Key features of Azure Machine Learning :-

① AutoML → Automated ML / Model selection / hyperparameter tuning and feature Engineering.
② ML pipelines → ML work-flows with reusable components
③ Compute Options → Azure VMs, GPU clusters, Integration and Kubernetes for scalable T & D.
④ MLOps → CI/CD / Monitoring / versioning.
⑤ Responsible AI → Interpretability

① Learn the Basics :-
→ Azure ML Studio
→ SDK (python based)
→ CLI

g concepts
→ workspaces
→ datasets
→ experiments

② Hands - on Practice :-
→ Azure ML Studio ⇒ Auto ML Experiments
→ Python - SDK ⇒ in Jupyter Notebook or Vs code
→ Run notebooks on Azure ML compute Instances.

③. Build and Deploy models:-

→ Train models using Sklearn, Tensorflow, (or) pytorch.

→ Deploy Models as Azure ML Endpoints using Azure Functions or AKS (Azure Kubernetes Service).

④ Explore MLOps & Advanced Topics:-

→ Learn Azure ML pipelines for model automation

→ Use MLFLOW

→ Implement model monitoring.

Agentic -AI → AI Agents

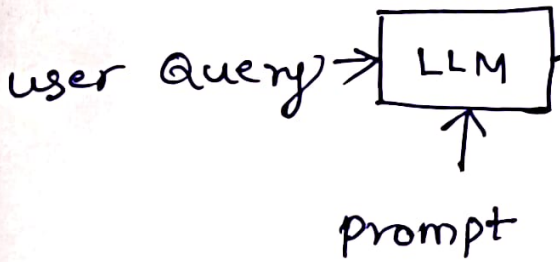
① What is Agentic AI

② How is different from Generative AI?

③ Frameworks used in to Create Agentic AI?

Generative AI:-

Create content → LLM Model



{ Generating Content }

Agentic AI:- Autonomous AI System

→ Autonomous AI System that can be able to perform their own task.

⇒ To achieve a specific goal it is the AI Syster, will be working independently without any human intervention.

Langgraph — Framework

→ It is a tool that helps to build AI workflows, where AI agents or steps work together in a structured way.

→ It uses graph like system to manage decision-making, memory and interactions.

→ It is useful for making smart chatbots, automation systems or AI assistants that needs to be remember things and make decisions.

→ In Generative AI the main aim is just to create the content.

Data → 100 log files or Pdf's or text docs.

Step 0: ↓ → reading the files from Azure Blob Storage.

Step 1: Data Extraction from log file or pdf

Step 2: chunking the data with the help of Recursive char Splitter.
(How will u decide the chunk size, how will u decide the chunk overlap?).

Step 3: Embedding the data. (open AI Embedding).

Step 4: Storing the Embedding in Vector DB (Azure Cosmos DB).

Prompt: Prompt Engineering completed.

LLM: For Providing content to the LLM,
Based on the Prompt instruction LLM will give answer.

without RAG.

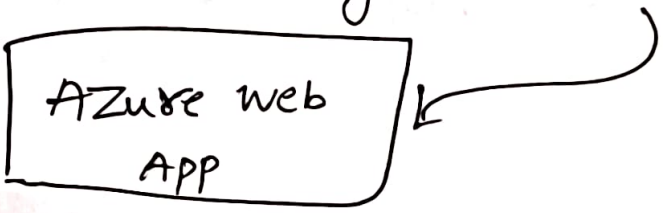
Deployment in AZure

→ CI/CD pipeline

→ Git hub actions

① Web app → Docker image ⇒ private image

② Docker Image → container Registry (AZure)



⊕ Container Registry

step 1 → Create Resource group - testdockerVenkatesh

Registry name - testdockerVenkatesh

Step 2 → Access key

Admin user → enabled

step 3 → Web app

Select different Resource group

Name - testdockervenkatesh

Docker Container

→ Central us

DOCKER → Single container/ AZure Container Registry

Registry - testdockerVenkatesh

Terminal :-

Creating the docker image → docker build -t testdockerVenkat
, azurecr.io/llmtest:late

Login into the docker → docker login testdockerVenkatesh.
azurecr.io

push the files into the docker → docker push testdocker
venkatesh
'azurecr.io/llmtest:lat

username :- testDockerVenkatesh

password :- from Azure Access Key

⇒ Creat web app

---

① Container Registry – Store dockerimage

② Azure Web app

1. What is Docker
2. What are Containers
3. Containers Vs Virtual Machine
4. Docker image Vs Container
5. Practical Implementation of Dockers.

Why Containers :-

Myself Venkatesh,

→ I do have Exposure overall 3.1 years of IT Experience.
→ Where I started my carrer as a data scientist.
→ My previous company is Accenture, for 3.1 years I worked over there.
→ When it comes to Machine Learning, DL and Gen AI I do have overall 3.1 years of Experience
→ So there I do have Exposure to ML, DL, Gen AI, NLP and at the same time → Connecting with the Client
→ taking the requirement from the Client
→ Doing the data collection -②
→ understanding the problem statement -①
→ Data preprocessing -③
→ Data cleaning / Manipulation / Feature Engineering -④
→ Model Building / Model Training / Model validating
→ checking the accuracy
→ Developing the production Ready code
→ pushing the code to the Code cloud Repositeries
→ from there maintaining the repositories over there
→ Maintaining the code in the feature Branches
→ from their integrating the feature Branches with the develop branches.

→ In Gen AI I do have exposure from the past 2 years, I have been working with Gen AI related projects.

→ When it comes to my specialization NLP is my strength and I am very good at gen AI dealing with Rag pipeline

→ And finally when it comes to cloud experience

→ I do have Exposure of using Azure Data-bricks

→ Azure ML → Azure function → Azure Blob Storage

→ Running the LLM Models from the Azure prospectus

→ Connecting Azure data bricks with the Azure functions, like that I do have Exposure till

→ deployment of the models also.

⇒ Deployment type :-

→ I do have Fast API and Flask API

Sometimes the frontend team will ask the predicti -ons in the Jabulay format and sometimes they will ask in Jason format

→ Based on their requirement I will be providing the results to them.

When it comes to the Azure data bricks deployment so whatever the main function, we are creating in the data bricks we will be calling that function and we will be executing.

→ once executed we will get job run Id
→ that job run Id will be passed to the Azure func frontend program.

→ Creating Github actions
→ Creating CI/CD pipelines
→ Dockers / Kubernetes that will be taken care by devops team.

project :-

(1)

1. Bais and Variance
2. Cross Validation and importance
3. Overfitting and underfitting
4. How to handle missing data
5. techniques used in feature selection
6. what is Recursive feature Elimination
7. Bagging Vs Boosting
8. Model performing well but not in production how to handle.
9. classification metrics when to use for which type of dataset
10. Different between llm and NLP
11. What is LLM
12. Differences between fine-tuning and prompt Engineering
13. NER in NLP?
14. Joins Concept in SQL.