What is RAG (Retrieval Augumented Generation)

is a method where an AI model first finds relevant information from a data sources (like documents) and then uses that information to give a betta and more accurate answer.

Lets Start Gen AI

Evolution of LLM's. Large Language Models :)

Can process sequences of any length ← used for handling sequential data

Simple RNN

- Struggle with long sentences, Vanishing gradient descent
- Loop architecture, O/p of one step is passed to input to another step

LSTM RNN

- Can remember long sequences, solves VGD
- Remembers information for longer time.
- uses gate architecture (input, forget, output)
- Slow to train, uses more computational power

GRU RNN

- Simplified version of LSTM
- fewer gates (reset and update gates)
- faster than LSTM
- may not as efficient as LSTM's for very complex sequences

Bidirectional RNN

- read a sequence from both forward and backward
- consider both past & future words to make predictions.
- Improve accuracy in many NLP tasks
- needs whole seq. before making a decision, computationally inefficient

Encoder-Decoder Architecture

- When the I/p, O/p are of diff length
- Language translation (english → French)
- text summarization (long article → short summary)
- handle variable length i/p, O/ps
- long sent, forgot imp details (bottle neck problem)
- training time q/p

Self attention

- focus on imp words in a sentence.
- checks all the words at once and gives high importance to relevant words.
- Remembers long sents better than RNN's
- uses lot of memory for long texts, Computationally inefficient.

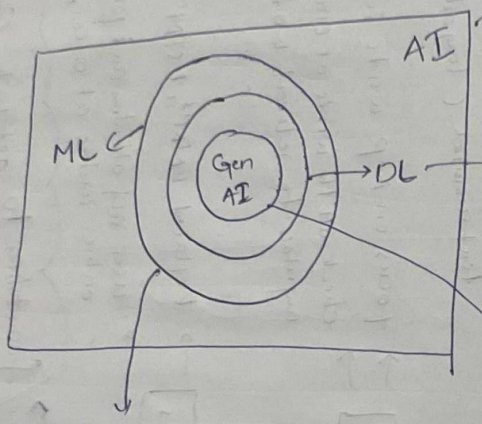
Transformers

- Replaced RNN's, LSTM's
- uses self attention and process the entire sentence at once., superfast
- extremely powerful for long texts,
- Can process entire input once (parallel processing)
- huge data to train well, Computationally expensive (powerful GPU's)

foundation to get into Gen AI and work with LLM models

Generative AI

AI Vs ML Vs DL Vs Gen AI



AI: Build application that can perform its own task without human interuption.

Netflix: recommendation system

self driving Car

Deep learning: is Built to mimic human brain.

We wanted machines to think how human think

DL (majorly we have these three)

- SLNN, MLNN, ANN
  - ML can be solved using ANN
- CNN & Object detection
- RNN & its variants.
  - Text related usecases & Time series usecases

Machine Learning

Statistical to perform statistical analysis, visualization, prediction and forecasting.

on Data → for understanding the data.

Generative AI: → models

- **Discriminative** → classification prediction
  - ↳ Trained on label datasets
- **Generative AI** → Generates new data, trained on huge data sets.
  - LLM (Text)
  - LIM (Images)

Large language Models (LLMs)

Companies are in the race, to make the best LLM model

Open AI → GPT 4 (gpt 5)
Meta → LLama2 (Llama3)
Google → Gemini (gemini pro)
Anthropic → Claude3

↑ Companies and their LLMs

LangChain is the framework, which we can able to work with all LLMs models

foundation models

also called as

pretrained models (because these are trained with huge amount of data)

↓ huge data

We can use these specific models for domain specific use cases also.

to use these pretrained models for domain specific use cases we should Finetune, it with custom dataset

↳ that is related to specific domain your are interested to work on.

How ChatGpt is trained

Internet Data
↓ (Website, article, Wikipedia, books, etc..)

Stage 1: Generative pre-Training

↓

Base GPT model

↓

Stage 2: Supervised fine tuning (SFT)

↓

Fine-tuned ChatGpt model

↓

Stage 3: Reinforcement Learning HF ← (human feedback)

↓

ChatGpt model.

Stage 1: Generative pre-training

Internet huge input text data

↓

Transformers

↓

Base GPT model

Tasks
- lang translation.
- Text summarization.
- text Completion
- Sentiment analysis

↓

What we want

↓

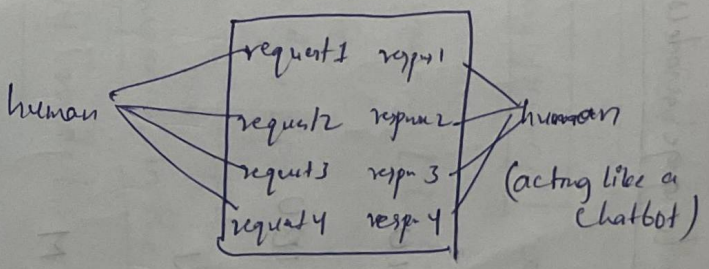
Conversation ChatBot

Stage 2: Supervised Fine tuning (SFT)

req ⇒ Conversation history

res ⇒ Best ideal response

Real Conversations.



Reinforcement Learning through human feedback

Stage 3 ⇐

☆☆☆☆☆

SFT ChatGpt → res / req ← human agent

- alternate responses.
- response ranking
- Ranked responses
- Reward model → (Context history: A, B, C, D → scores A, B, C)
- proximal policy optimization (PPO)

SFT training Data Corpus

req	res
req	res
req	res

millions of records

⇒ Training → Base GPT model → SGD optimizer ⇒ SFT ChatGpt model

Evolution of Large language models (LLMs)

- 1967 → Eliza
- 1970 → SHRDLU
- 1980 → XCALIBU
- 1998 → RNN
- 1997 → LSTM
- 2017 → Transformers
- 2018 → Bert, GPT
- 2019 → GPT-2, RoBerta, XLNet
- 2020 → GPT-3 (OpenAI)
- 2021 → GPT-3.5
- 2022 → paLM, InstructGpt, ChatGpt
- 2023 → Llama, Gpt-4, Bard ---
- 2024 → OpenAI → multimodel (text, image) → GPT 4.0
  - → Gemini pro 1.5
  - → Llama3, anthropic, Claude

Lot of research happening in the recent years.

LLM model Analysis

artificialanalysis.ai

↓

Comparing various LLM's that are present right now.

(Quality, Speed, price)

Major Companies, in the Competition to build to best LLM model

1. open AI
2. Google
3. Meta
4. MishalAI
5. Anthropic
6. Cohere
7. Databricks
8. Reka
9. AI21 Labs
10. Deep seek
11. snowflow
12. 01.AI

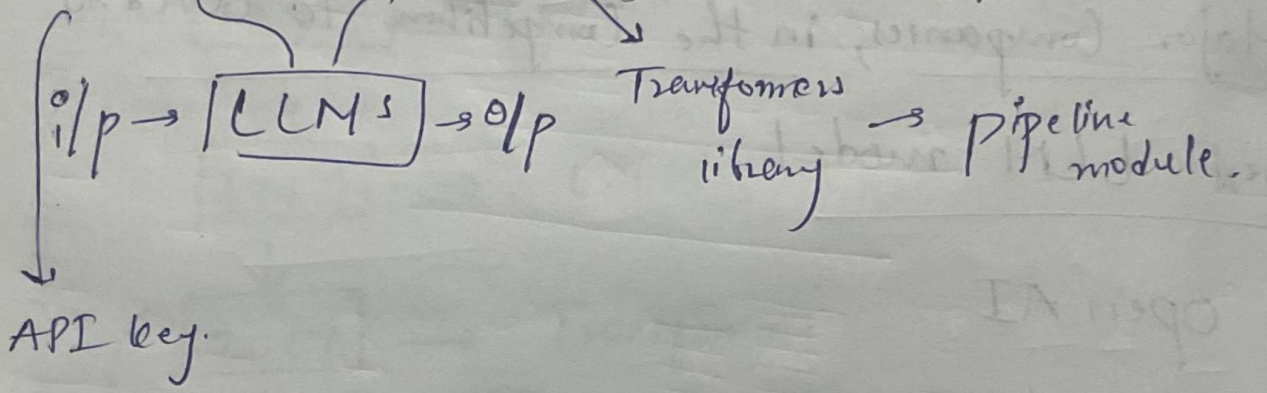
Langchain Ecosystem.

Langchain right now is most common framework, that is specifically used to build gen ai applications.

① Why Langchain

② Langchain ecosystem.

OpenAI, Huggingface. → launched popular LLM's.

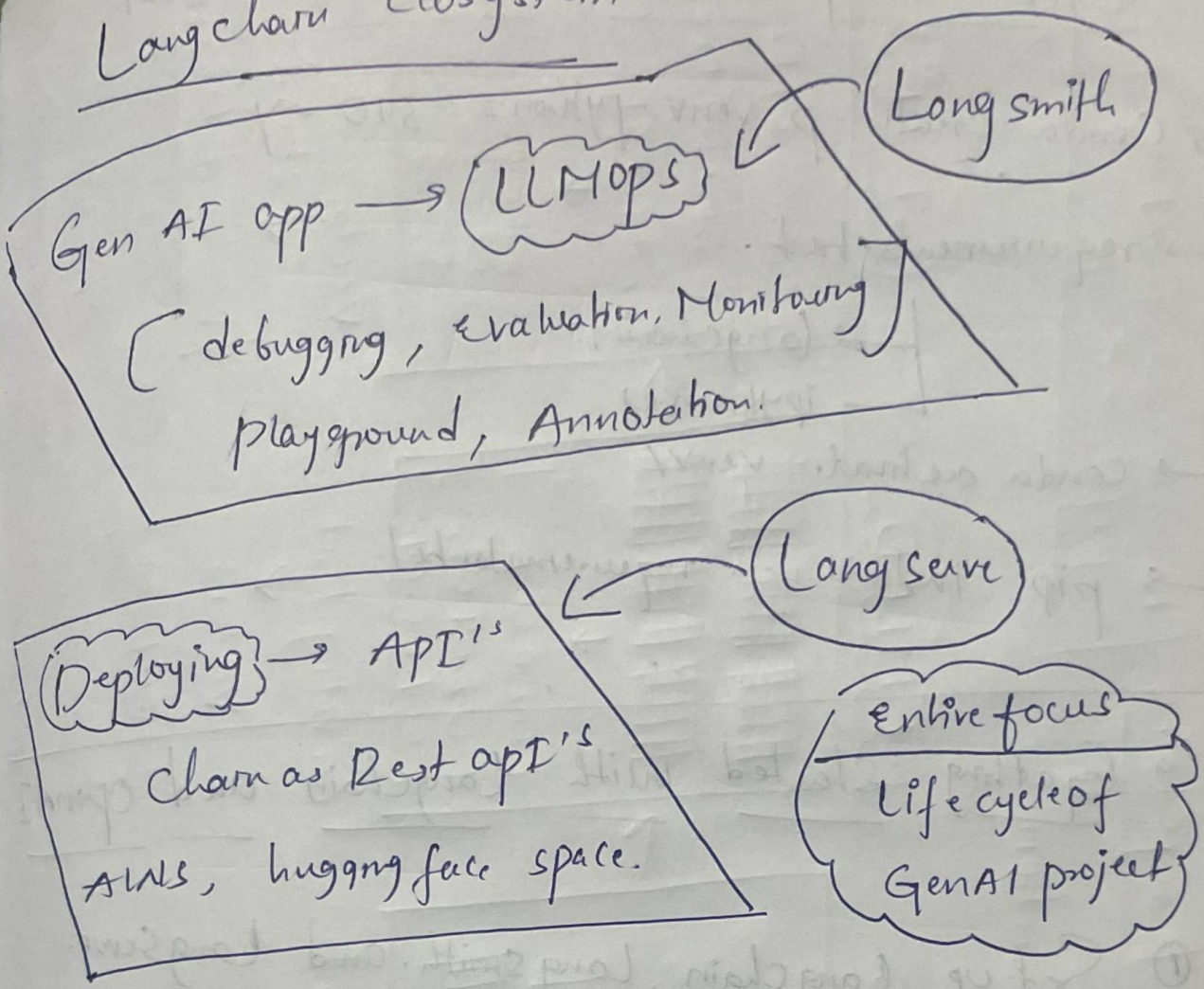


Similarly we have Meta, Google etc. for accessing each companies model (there is going to be an different library).

Langchain: (open source

↳ provided a common framework for accessing all LLM's models of different companies and for developing gen ai applications

Langchain Ecosystem.



Langchain → cognitive architectures

Chains agents Retrival strategies.

Langchain Communities → Integration Components.

Model I/O
model, prompt,
Example selector
output parser

Retrival
Retriver,
Docu loader
vector store
text splitter
embedding model

agent Tooling
Tool
toolkit

Langchain Core → protocols.

LCEL - Langchain Expression Language.

Parallelization Fallbacks Tracing Batching Streaming Async Composition

CREATING VIRTUAL ENVIRONMENT

→ Conda create -p venv python==3.10 -y.

→ requirements.txt

- langchain
- ipykernel

→ conda activate venv/

→ pip install -r requirements.txt

Getting Started with Langchain and OpenAI

① Set up Langchain, Langsmith, and Langserve.

② Use basic and common components of langchain: prompt templates, models and output parser

③ Build simple application with langchain

④ Trace your application with Langsmith

⑤ Serve your app with Langserve.

Create langchain api key
openai api key
.env file

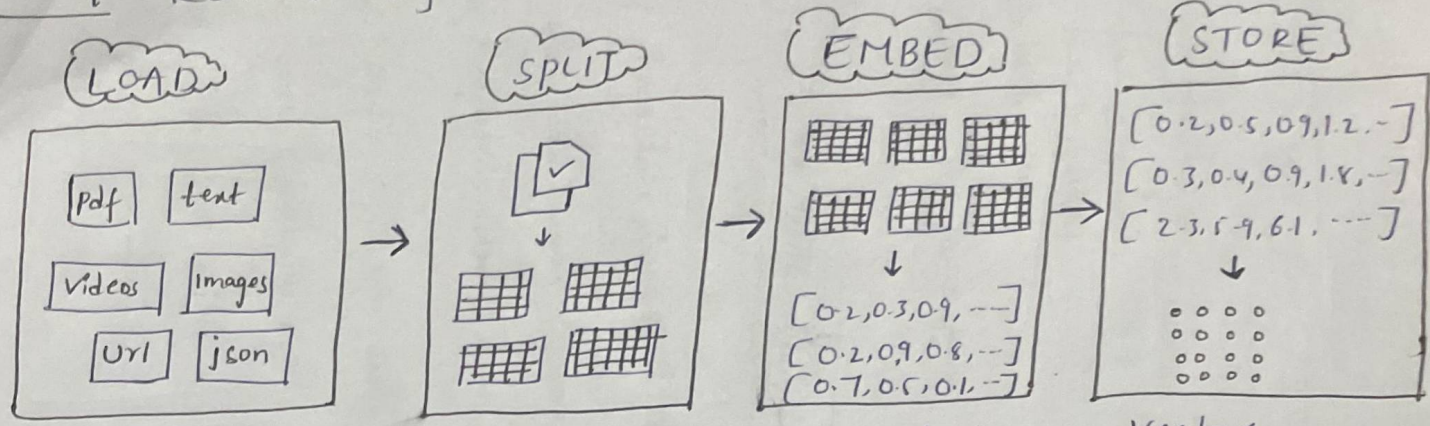
requirements.txt : python-dotenv → to import .env file into code.

Important Components of Langchain

Datasource pdf's → GenAI app ← query

RAG.

① RAG: Retrival Augmented Genration.



Loading the data from various data sources

Data Ingestion

①

split the data into Chunks

Data → Chunks

LLM models → Limitation: Content size

Convuling text → Vectors

Similarity search algo,

Cosine search

Diff LLMs has diff embedding teqnes.

Vectors → stored

Vector database

④

① FAISS

② Chroma DB

③ ASTRA DS

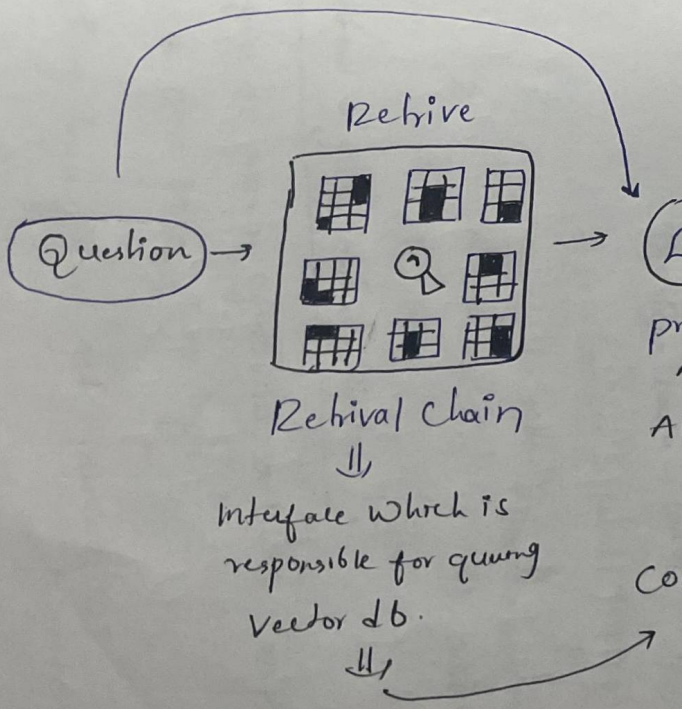
Module 1

Data Transformation

②

Embedding

③



⇒ O/p response.

Retrival chain

Interface which is responsible for quering Vector db.

AI assistant + Content Info

Exploring each step in detail and implementing throug Langchain

Module 2

① Document Loader in Langchain

↓

are used to import data from diffent sources into a format Langchain can work with.

text, docs, websites, pdf etc.

from Langchain_community.document_loader import

TextLoader → loads "text"

PyPDFLoader → loads "pdfs"

WebBaseLoader → loads "webpages"

ArxivLoader → loads → research papers

WikipediaLoader → loads → "wikipedia articles"

Content size (Content window):

→ refers to max no of tokens a LLM can process at once in a single input.

→ anything beyond that get truncated (or) forgotten

→ **i/p prompt + response = Total tokens**

↓↓

Content size.

② Splitting the documents

① Recursive text splitter:

Splits nicely first by paragraphs, then by Sentences, then by words, until each piece is the right size.

② Character Textsplitter:

Splits based on fixed character count.

③ HTMLTextSplitter:

Helps to split the HTML documents into clean text chunks while ignoring HTML tags and keeping meaningful content.

④ Recursive json splitter:

used to split large json documents into smaller meaningful chunks.

③ Langchain Embeddings

Embedding:

Converts text (chunks) to Vectors

↳ There are different LLM models which provides Embedding model like.

① Open AI
② Ollama
③ Hugging face
④ Claude 3
⑤ anthropic.

→ Here we are going to focus on

① Open AI (paid)
② Ollama (open source)
③ hugging face. (open source)

these three provides good accuracy which is more than sufficient.

① Open AI Embeddings

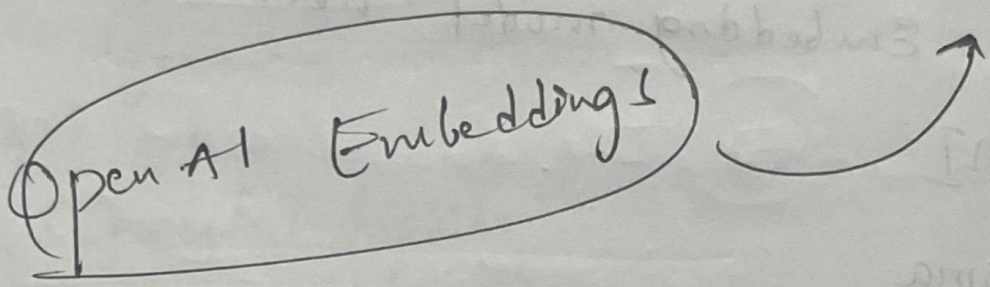
→ open ai api key (5 usd)

→ billings

→ Create new Secreate key.

→ Openai offers three popular Embedding models for converting text to Vectors

① text-embedding-3-large most capable one
② text-embedding-3-small smaller embedding mod
③ text-embedding-ada-002 the older one.



② OLlama Embeddings

Ollama is a platform where you run open-source models like Llama (meta), Mrshal (mrshal AI), Gemma (Google), Code LLaMA (meta), Neural Chat (Intel), and Phi (microsoft) locally on your computer for privacy, offline access and easy AI Experimentation.

OLlama embeddings

Using open source models
Llama, Gemma, deepseek-R1 etc..

Using Embedding models
→ mxbai-Embed-large
→ nomic-Embed-text
→ all-minilm

⑧ Huggingface Embeddings

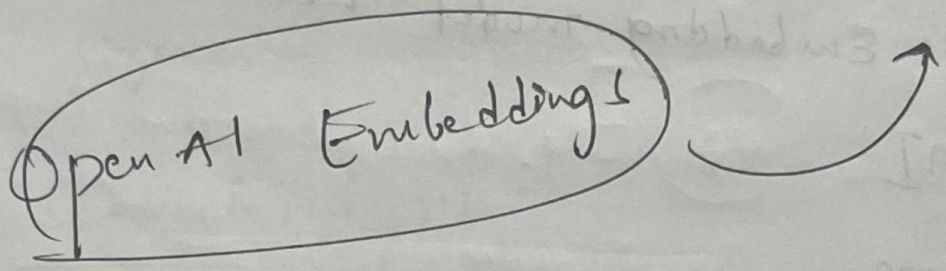
Sentence transformers (library)

```
from langchain_huggingface import
                    HuggingFaceEmbeddings
```

① all-MiniLM-L6-V2
② all-mpnet-base-V2
③ paraphrase-MiniLM-L6-V2
and many more ...

→ Open ai offers three popular embedding models for converting text to vectors

① text-embedding-3-large most capable one
② text-embedding-3-small smaller embedding model
③ text-embedding-ada-002 the older one.

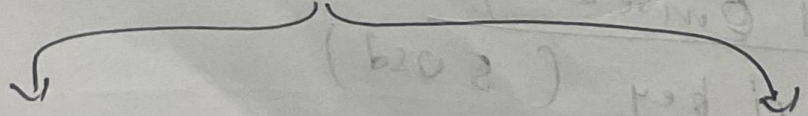


---

② OLlama Embeddings

Ollama is a platform where you run open-source models like Llama (meta), Mrshal (mrshal AI), Gemma (Google), Code LLaMA (meta), NeuralChat (Intel), and phi (microsoft) locally on your computer for privacy, offline access and easy AI Experimentation.

OLlama embeddings



Using open source models

Llama, Gemma, deepseek-R1 etc..

Using Embedding models

→ mubai-embed-large
→ nomic-embed-text
→ all-minilm

⑧ Huggingface Embeddings

Sentence transformers (library)

from langchain_huggingface import HuggingFaceEmbeddings

1. all-MiniLM-L6-V2
2. all-mpnet-base-V2
3. paraphrase-MiniLM-L6-V2

and many more ...

④ Vector Store DB's

in lang chain are tools used to store and sear embeddings - espeacially useful in Rehival-Augmented Generation (RAG) systems.

→ There are many vector store Db, here are few

① Faiss
② Chroma db
③ Asha db
④ pinecone.

① **Faiss: Facebook AI Similarity Search** is a library for efficient similarity search and clustering of dense vectors, mainly used to quickly find similar items (like text or images) based on their vector embeddings

library.

faiss-cpu

```
from langchain_community. import
```

While working with LLM'' models, We cannot use Vector store db directly instead we Convert them into Retrivers and give to LLM models

```
retrivel = db.as_retrivel()
```

Similarly we have

↳ Similarly Search store db.

↳ Similarly_Search by_vector

↳ Storing vector store db in local

```
db.save_local("faiss_db")
```

↳ loading Faiss locally

```
newdb = Faiss.load_local(',2,3)
docs = newdb.Smrly_Se(query).
```

Libry.

faiss-cpu

② Chromadb

Chromadb (or) library.

↳ open source vector database

langchain_chroma

Building Important components of langchain

1. Set up LangChain, Langsmith, Langserve
2. using basic common components like.
   prompt templates,
   models,
   output parsers
3. Building a simple application with Langchain.
4. Trace your applications with Langsmith.
5. Serve your applications with Langserve.

Project 1

Building GenAI app using Langchain with OpenAI

↓

Querydocs AI

↓

Answers your questions from group of documents

↓

model name is "gpt-4o"

Steps involved in making project

Step 1:

Setting up environment and Api keys

↳ Load Api keys from .env file

Step 2:

Load Data from Website

↳ WebBaseLoader

Step 3:

Load website data as a Document

↳ loader.load()

step 4:

Split documents into Smaller Chunks

↳ RecursiveCharacterTextSplitter()

step 5:

Convert text to vectors (Embeddings)

↳ OpenAIEmbeddings()

step 6:

Store Vectors in a Vector Databases (Faiss, Chroma ..)

↳ FAISS, CHROMA, ..

step 7:

Search relevant chunks from Vector store.

↳ SimilaritySearch()

step 8:

Create the LLM model (GPT-4o)

↳ ChatOpenAI

step 9:

Create the prompt format (ChatPromptTemplate)

↳

ChatPromptTemplate.from_template("""

""")

10 Step:

Create the document chain

↳ create_stuff_documents_chain(llm, prompt)

→ Combines model & prompt into a chain

→ takes multiple docs + prompt → model → answer

Step 11:

Try Chain with manual content

↳ manually giving the document and checking

Step 12:

Turn Vector Store into Retriver

→ vdb.as_retriver()

→ can auto pick the best matching documents

Step 13:

Create a Full Retrieval Chain.

Retriver → to find relevent docs

document_chain → takes all docs adds prompt and gives to LLM

Step 14:

Ask a Real Question

↳ Check for new questn

step 15:

View the Retrived Documents.

```
resp = retrivel_chain.invoke({'input': ...})
resp['answer']
```

project-2

Building Gen AI App using Langchain with Ollama

↓ Ollama Q&A Assistant

↓ model is "deepseek r1"

Step1: Imports - Setting up the Environment.

↳ libraries, .env file

Step2: Langchain and Ollama imports

→ Ollama, Chatprompttemplate, Stroutputparser.

Step3: prompt template Creation

→ helps in controlling how AI should respond

Step4: Building the Streamlit UI

→ for building UI

Step5: LLM setup and chain creation

→ prompt | LLm | output-parser

Step6: Handling User Input and Showing O/P.

↳ asking Que from user and answering back.

This run locally using deepseek r1 via Ollama.

Building Basic LLM application using LCEL

LangChain Expression Language

Getting started with open source Models using Groq Api

① Open AI Api key → paid

② Open source models
   ① Llama3
   ② Gemma2
   ③ mistral
   } through platform Groq → similar to Ollama

Groq
→ low latency
→ Cloud host
→ LPU
language processing unit

Create Groq Api key from Groq Cloud
↓
helps in calling opensource models.

Why Groq

AI inference

↓

process of using a trained AI model to make predictions or generate O/ps from new input data.

Training → model learns from data

Inference

When the model uses, what it learned ~~from data~~ to give answers

Que → ChatGpt reply → Inference

Google translate converts Sentence → Inference

AI models detect faces in photos → Inference

Why fast Inference matters.

① Real time Experience

slow AI → Frustating

fast AI → Seamless Experience

② Better Users engagement

③ Useful in Critical applications

healthcare, finance, security ...

④ Cost efficiency at scale

less compute time / request

project-3

Building Basic LLM application using LCEL

↓

Using Groq
Gemma2-9b-It

↓

Langchain Expression language

Steps involved in project

Step 1: Importing required libraries

fastapi , StrOutputParser , OS , add_routes
ChatPromptTemplate ChatGroq , dotenv

Step 2: load api key from .env file.

→ load_dotenv()
→ Groq_api_key

Step 3: Initialize Groq LLM model

ChatGroq (Gemma2 -9b-It)

Step 4: Define prompt Template.

↳ LLM understand its role & task clearly

System prompt , User prompt

Step 5: Define Output parser

↳ converts LLM o/p into plain string format

Step 6: Build the Langchain Chain

Building chain → pipeline

prompt template | model | parser.

Step 7: Create FastAPI App.

→ creates app

→ allows people to send (or) get
request response.

Step 8: Expose Langchain Chain as API Route

→ connects langchain model to API

Step 9: Run the APP with Uvicorn

→ starts the web server that will run your Fast api app on your Computer (localhost)

Simple translation API that runs on your computer using FastAPI. It sends your input text to Groq model online and returns the translated sentence

Langchain & Langserve helps to connect requests to AI smoothly.

project - 4

Building Chatbots with Conversation history using Langchain.

Load Credentials from .env

os.getenv(), api keys

Run Inference using Groq LLM

Model, ChatGroq

Chat without Memory

Stateless chat, model.invoke(...)

Keep history per session

Memory store, ChatMessageHistory

Add memory to model

Memory wrapper, RunnableWithMessageHistory

Identify Each user's history

Session ID's, session_id = "chat1"

Define Structured Inputs.

prompt templates, ChatPromptTemplates

trim messages

TokenTrimming, avoid Exceeding token limits

Combine trimming, templating, inference

full chain, RunnablePassthrough → prompt → model.

Important functions in Langchain, that everyone should know before Creating Chatbots with Conversation history.

① ChatMessageHistory( )

Stores all messages in a single chat

② get_session_history(session_id)

gets message history for session (s) Creats new one if it does not exist

③ RunnableWithMessageHistory( )

adds memory support to your chatbot

④ ChatPromptTemplate.from_messages([...])

Builds a prompt using system and user messages

⑤ MessagePlaceholder(var_name = "messages")

Reserves space in the prompt to insert past messages

⑥ trim_messages(---)

removes older messages (s) unimportant ones

⑦ itemgetter("messages")

Pulls out the messages key from dictionary

⑧ RunnablePassthrough(...)

prepares data (like trimming msgs) before sending to the model

project-5

Conversational Q&A chatbot with Message History [Full steps]

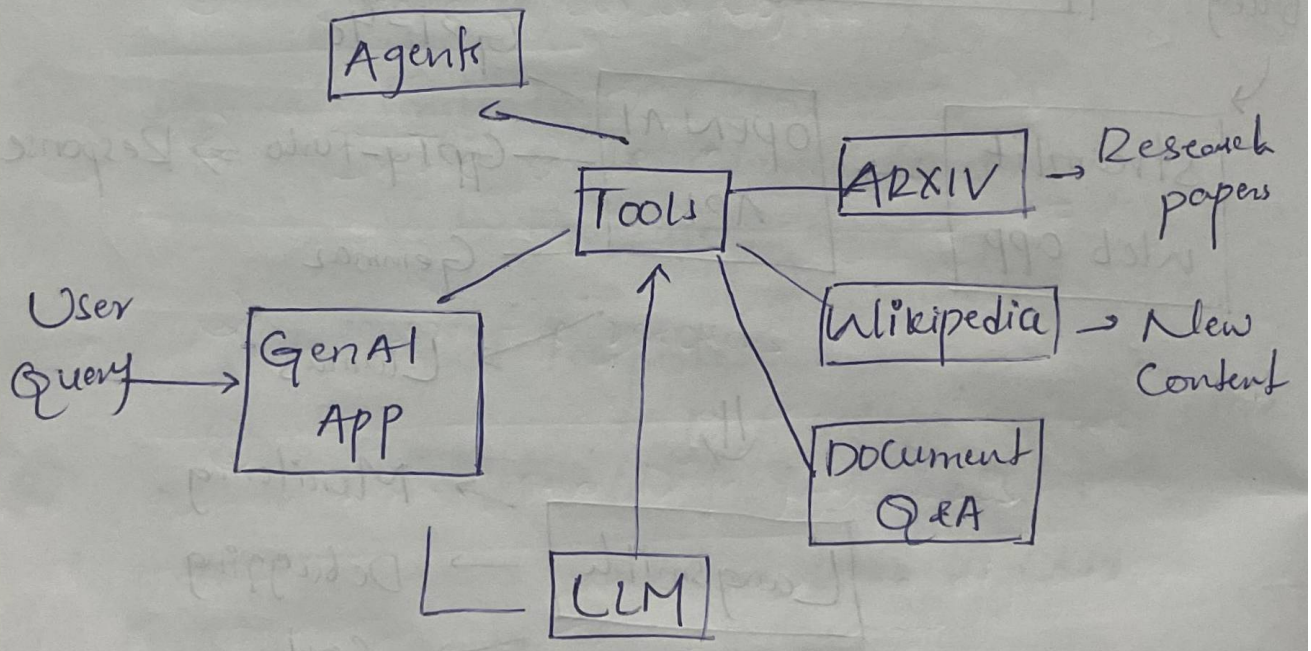
1. Setup and Load environment
   ↳ Loading API keys
2. Initialize LLM → Loading Model
3. Load Data from Website → Scraping the data
4. Split Text into Chunks → Breaking long text → chunks
5. Generate Embeddings → Chunks to dense numeric Vectors
6. Store in Vector Database → Store embeddings
7. Create prompt templates → Define how System & User interact
8. Create Retrieval Chain
   → Retrieve relevant chunks and pass to LLM
9. Ask Questions using the Chain → ask ques, get context based answers
10. Enable Chat History Awareness
    Add memory to prompts, make retriever aware of flow
11. Manage Chat History
    Keep track of human and AI messages
12. Ask follow-up Questions.
    → play with chatbot

Search Engine With Tools And Agents

Tools: tool is a function that does a specific task, like doing math, searching the web (or) running code. It helps the chatbot get things done.

AGents: is a smart controller that decides which tool to use when, in order to answer the user question.

OpenAI LLM model → Gpt-4o ⇒ December 2023 → Data.



End to End Search Engine

Wrapper: Gets data from website.

QueryRun: Uses the Wrapper to answer your question.

Steps to Create an Agent

1. Creating Wrappers
   → WikipediaAPIWrapper, ArxivAPIWrapper etc...
2. Creating tools using those Wrappers
   → ArxivQueryRun, WikipedraQueryRun.
3. Combining all the tools
   → add all tools
4. Creating LLM
   → Ollama, openai, Groq ...
5. Create prompt Template.
   → langchain.hub.pull("")
6. Create Agent
   → langchain.agents.create_openai_tools_agent.
7. Create Agent Executor.
   → Langchain.agents.AgentExecutor
8. invoke Agent Executor with different questions

→ agent.executor.invoke({"input":"What is AI"})

project 7: Chat with SQL DB with langchain

SQL Toolkit and Agenttype

① preparing the data for SQLlite3 database

① Import the SQLlite Library
→ import sqlite3 (python library)

② Connect to the Database
→ Creates a new database

③ Create a cursor object
→ used to Execute SQl commands

④ Create a Table
→ Define table

⑤ Insert the records
→ add records to table

⑥ Fetch and Display Records
→ Retrives and displays all the records

⑦ Commit the changes
→ Save all changes made to db

⑧ Close the Connection.
→ Close to free resources.

Steps involved in project

1. Import libraries — ChatGroq
   streamlit, pathlib, langchain, sqlalchemy, sqlite3,
2. page setup
   title, headings
3. Database selection
   sqlite3 local, MYSQL external.
4. Collect MYSQL details if selected
   username, password, DB name.
5. Enter Groq API key
   Groq api key
6. Set up the LLM
   initialize LLM
7. Configure the database.
   sqlite (or) MYSQL (create_engine)
8. Connect to selected DB
   User choice.
9. prepare SQL toolkit
   creating tool
10. Create LangChain SQL agent.
    Creating agent
11. Session history initialization
    → Starts new chat history
12. Show Chat history
    → Display messages
13. User sends a New Question
    → user input.
14. Run Query and Show Respons.

project 8: Text Summarization with Langchain

① When you want to Summarize small number of tokens: Fits in LLM context Window

→ Manual prompting

→ prompt template with LLM chain.

① **Manual prompting**

↳ using AIMessage
SystemMessage
HumanMessage
} = msgs

→ llm(msg) simple ✓

② **prompt template with LLMChain**

modules → LLMChain, PromptTemplate

prompt template

→ input_variables = [ ]

→ template =
generic
prompt template

LLMChain

→ llm = llm

→ prompt = prompt

Steps involved in project

1. Import libraries — ChatGroq
   Streamlit, pathlib, langchain, sqlalchemy, sqlite3,
2. page setup
   title, headings
3. Database selection
   sqlite3 local, MYSQL external.
4. Collect MYSQL details if selected
   username, password, DB name.
5. Enter Groq API key
   Groq api key
6. Set up the LLM
   initialize LLM
7. Configure the database.
   sqlite (@) MYSQL (create_engine)
8. Connect to selected DB
   User choice.
9. prepare SQL toolkit
   creating tool
10. Create LangChain SQL agent.
    Creating agent
11. Session history initialization
    → Starts new chat history
12. Show Chat history
    → Display messages
13. User sends a New Question
    → User input
14. Run Query and Show Respons.

project 8: Text Summarization with Langchain

① When you want to Summarize small number of tokens: Fits in LLM context Window

→ Manual prompting

→ prompt template with LLM chain.

① **Manual prompting**

↳ using AIMessage, SystemMessage, HumanMessage } = msgs

→ llm(msg) simple ✓

② **prompt template with LLMchain**

modules → LLMChain, PromptTemplate

prompt template

→ input_variables = [ ]

→ template = genera prompt templat

LLMChain

→ llm = llm

→ prompt = prompt

② When you want to Summarize large number of tokens: does not fits in LLM content window

① Stuff Document Chain Summarization.

② Map Reduce Summarization Technique

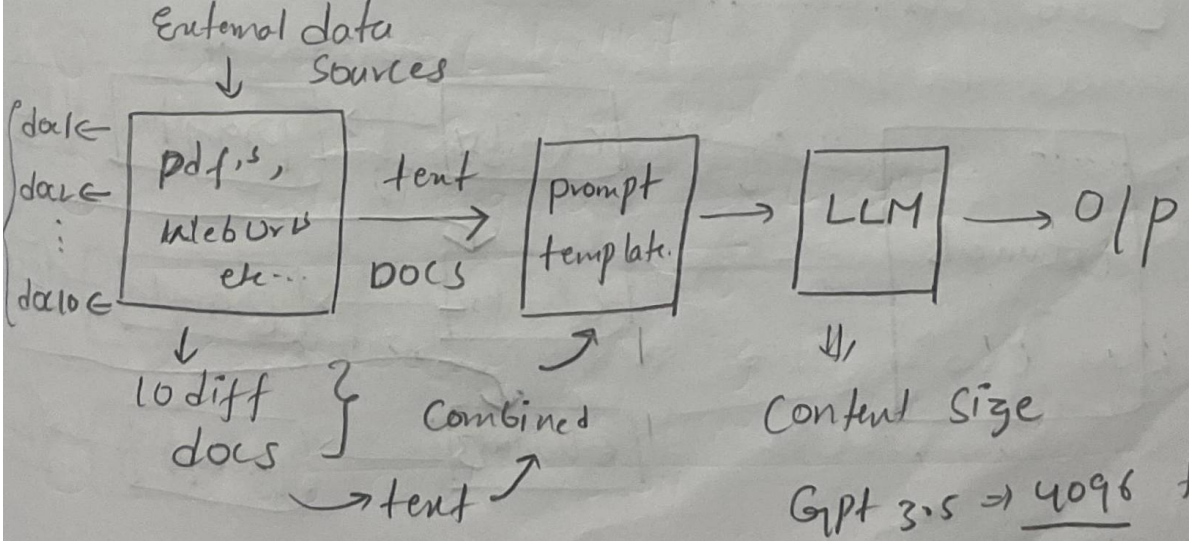
↳ Larger files

↙ Single prompt template

↘ Multiple prompt

③ Refine Chine Summarization.

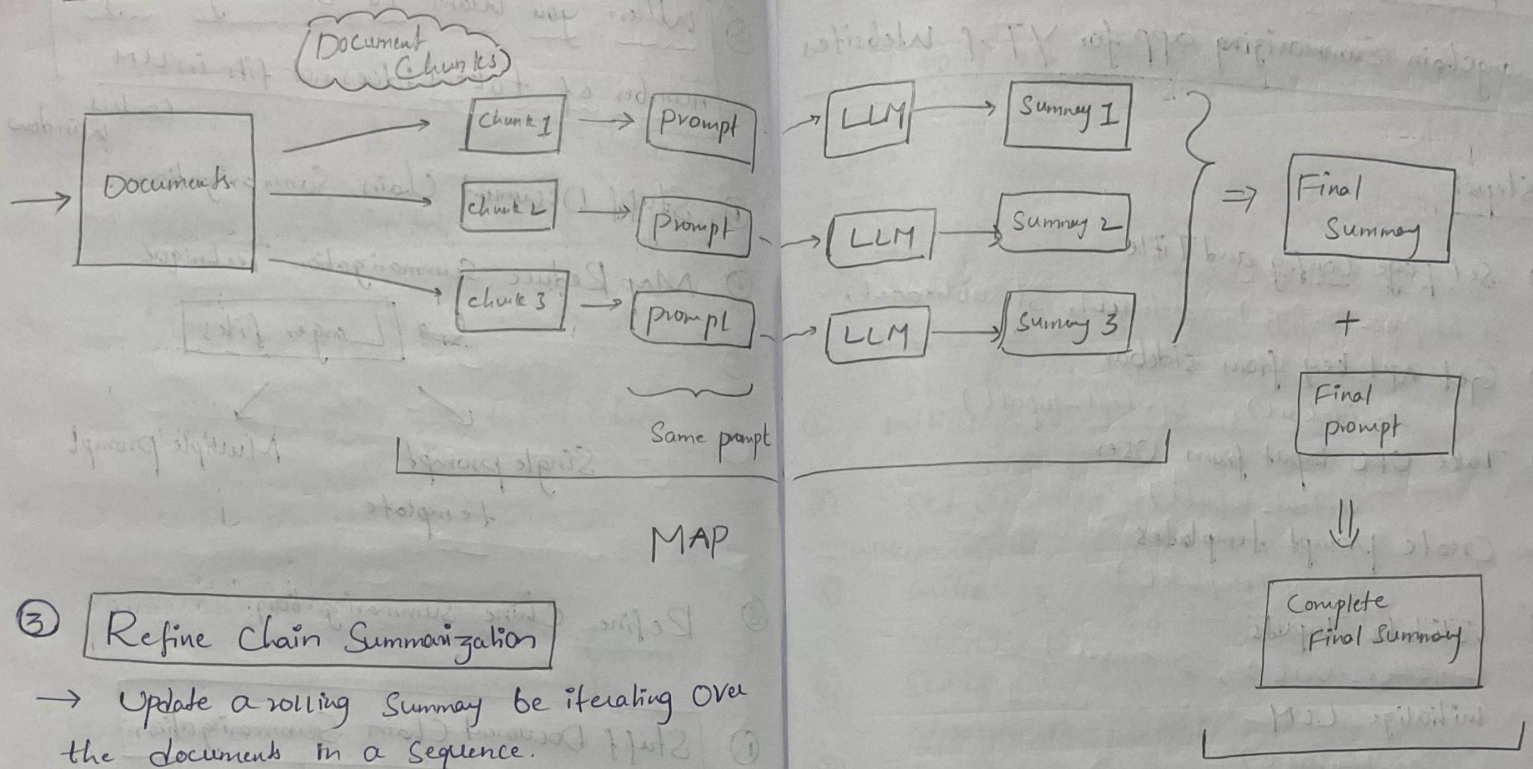
① Stuff Document Chain Summarization.



Challenges:

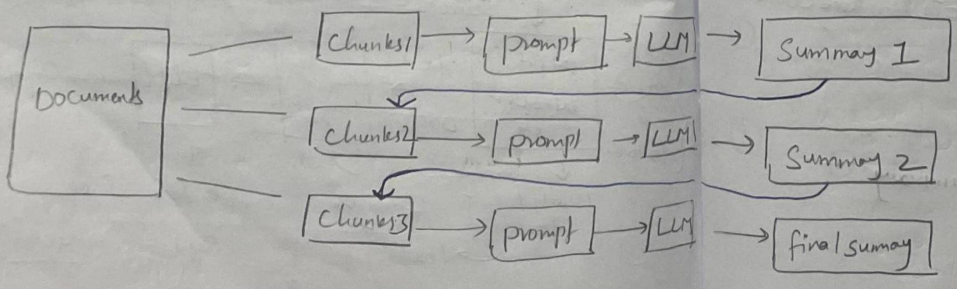
Larger documents ✗

② Map Reduce Summarization Technique



③ Refine Chain Summarization

→ Update a rolling summary be iterating over the documents in a sequence.



rolling Summary

Project 10:

Langchain Summarizing app for YT & Website

Steps:

① Set page Config and Title
St.set_page_config(), st.title(), st.subheader()

② Get API key from sidebar
st.sidebar(), st.text_input()

③ Take URL input from User
st.text_input()

④ Create prompt templates
PromptTemplate()

⑤ Validate inputs
validators.url(), st.error()

⑥ Initialize LLM
ChatGroq()

⑦ Load Content from URL
YoutubeLoader.from_youtube_url(), UnstructuredLoader()

⑧ Create Summarization Chain
load_summarize_chain( )

⑨ Run Chain and display Summary
chain.run(), st.success()

⑩ Handle errors.
try-except(), st.error()

Project 11:

Text to Math problem Solver Using Langchain

Steps:

1. Set up page config and title
   st.set_page_config(), st.title()
2. Enter Groq API key
   st.sidebar.text_input()
3. Initialize LLM
   ChatGroq()
4. Set up tools
   Wikipedia tool, Calculator tool, Reasoning tool
5. Initialize agent with tools
   initialize_agent (tools=[])
6. Set up chat history
   st.session_state(), st.chat_message()
7. User input section
   st.text_area()
8. Solve Button logic
   input check, append user Messages,
   agent Response,
9. Display final answer.

   st.chat_message("assistant").write(response)

   st.success()

85% of Course Covers:

- Gen ai and LLM models
- langchain for GenAI
- langchain with OpenAI
- Components and Modules in langchain
- OpenAI & Ollama.
- LCEL
- more than 12+ projects covered

  - Datastax Astra dB
  - how to deploy gen AI app^n using Streamlit

Project 12:

PDF Query RAG with Langchain & AstraDB

large, fast, high scalability, cloud based

AstraDB is a ready to use version of **Cassandra** made by **DataStax**, that runs in the Cloud

Steps:

1. Install and Import required Libraries
   pypdf2, cassandra, VectorStoreWrapper, OpenAI, Cassio, OpenAIEmbeddings etc...
2. Load apikeys and connect to AsbaDB
   openaiapikey, ASTRA_DB_APPLICATION_TOKEN, ASTRA_DB_ID, Cassio.init()
3. Read text from the pdf file
   Read the data
4. Split the text into Chunks
   CharacterTextSplitter()
5. Set up LLM and embeddings
   OpenAI(), OpenAIEmbeddings()
6. Create Vector store backed by Astradb and Insert Chunks
   Cassandra(), vector_store.add_texts()
7. Wrap the Vector store for Easy querying
   VectorStoreIndexWrapper()
8. Ask queries
   asha_vector_index.query(question="")

How to deploy Gen AI app on Streamlit Cloud

prerequisites:

1. Github account
2. Streamlit Cloud account (Linked to Github)
3. A working Streamlit python app.

Steps:

1. Create your Streamlit app.
2. create requirements.txt
3. push your project to Github
4. Deploy app on Streamlit Cloud.
   - → visit Streamlit Cloud
   - → Sign in with Github
   - → Click + New app
   - → branch → main, filepath → app.py
   - → Click deploy
5. Optional
   - → add configurations, add secreat for api keys
6. Share the app.

Deployment of Gen AI app in Hugging face spaces :

prerequisite:

① Github account

② huggingface.co account.

steps:

① Create new file in the github repository

.github / workflows / main.yaml

② Copy the code from "huggingface spaces github actions" and past in "main.yaml"

③ Now go back to hugging face spaces,

Create new space:

→ spacename

→ license.

→ streamlit

→ cpu basic 2vcpu free

→ public

→ Create space.

④ Create access token

→ Go to huggingface.co

→ access token

→ new token

→ Generate token (write mode)

⑤ replace your username in main.yaml to your hugging face username.

→ similarly replace spacename with the spaces name you gave before

⑥ Now go to github settings

→ Secrets and Variables

→ Actions

→ New repository Secreat

↳ Name HF_TOKEN

↳ Secret

↑

paste the hugging face access token here.

→ add secret

NVIDIA NIM

NVIDIA inference microservice is a prebuit, optimized container that delivers fast and scalable AI models inference with minimal setup.

project 13:

Building RAG Document Q&A with Nvidia NIM And Langchain.

Steps:

① Import Required libraries:

st, os, dotenv, Nvidia embeddings, Chat Nvidia, PyPDF DirectoryLoader, RecursiveCharacterTextSplitter, FAISS, ChatPromptTemplate, Create_retrival Chain, Stuff document chain.

② Load Api key:

NVIDIA_API_KEY = os.getenv(" ")

3. Initialize Nvidia LLM

ChatNvidia( ) ⇒ Namas-70b-Instruct

4. Create Vector Embeddings from pdf:

NvidiaEmbeddings( ), PyPDFDirectoryReader( ),

RecursiveCharacterTextSplitter( ).

5. Streamlit User Interface:

title( ), write, button

6. prompt template:

ChatPromptTemplate.from_template( )

7. Take User question and Get answer.

User input → retrieve relevant documents →

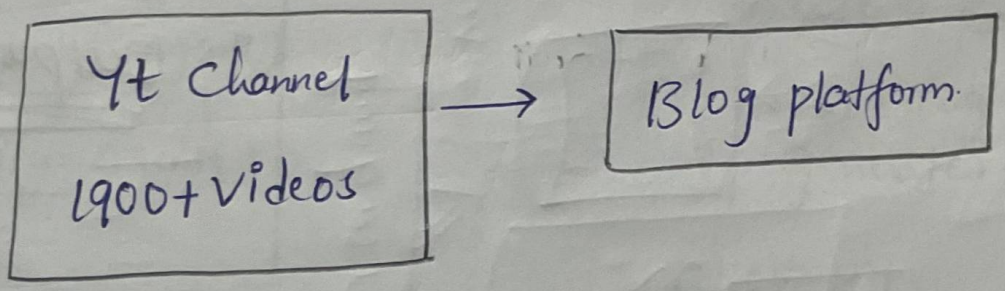
combine document with prompt → send to NvidiaLLM

→ generate the answer.

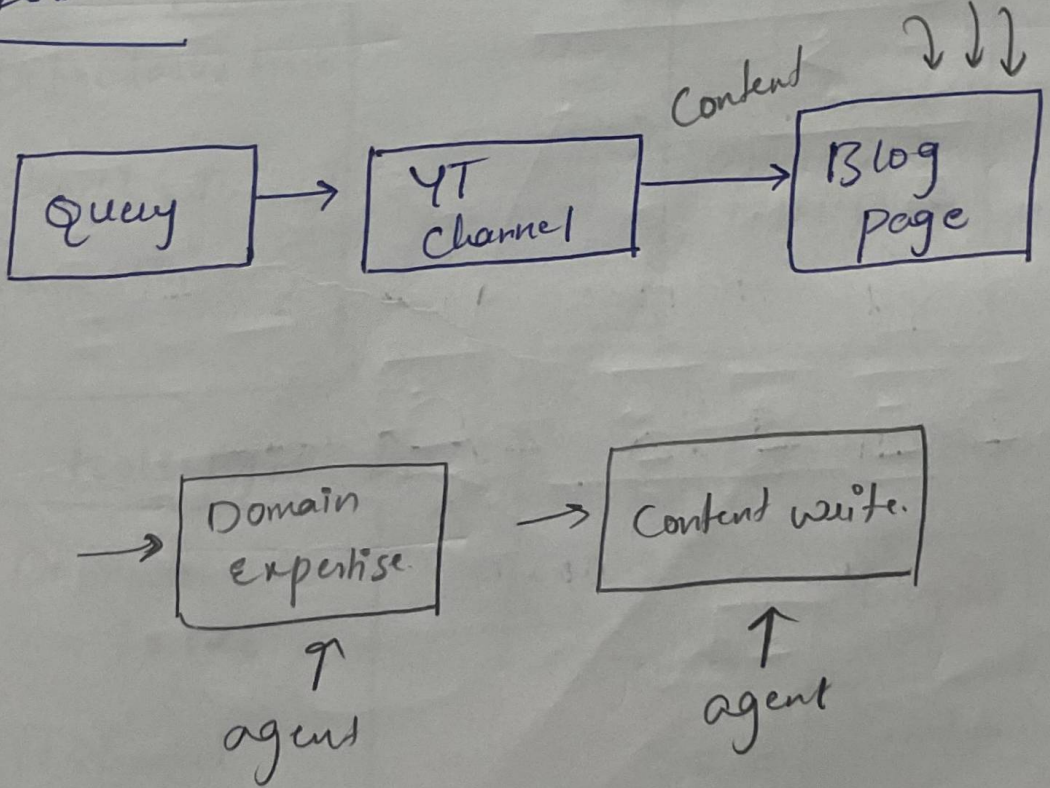
CREW AI

Crew AI is a framework that allows multiple AI agents to work together as a team to complete complex tasks efficiently through collaboration and coordination.

project:



CREW AI



CREW AI

3rd party apis, that helps you to complete task

AGents

Tasks

TOOLS

Researcher → Explore Videos Yt Channel → YT Tools

Content writer → Based on research. → Blog pages

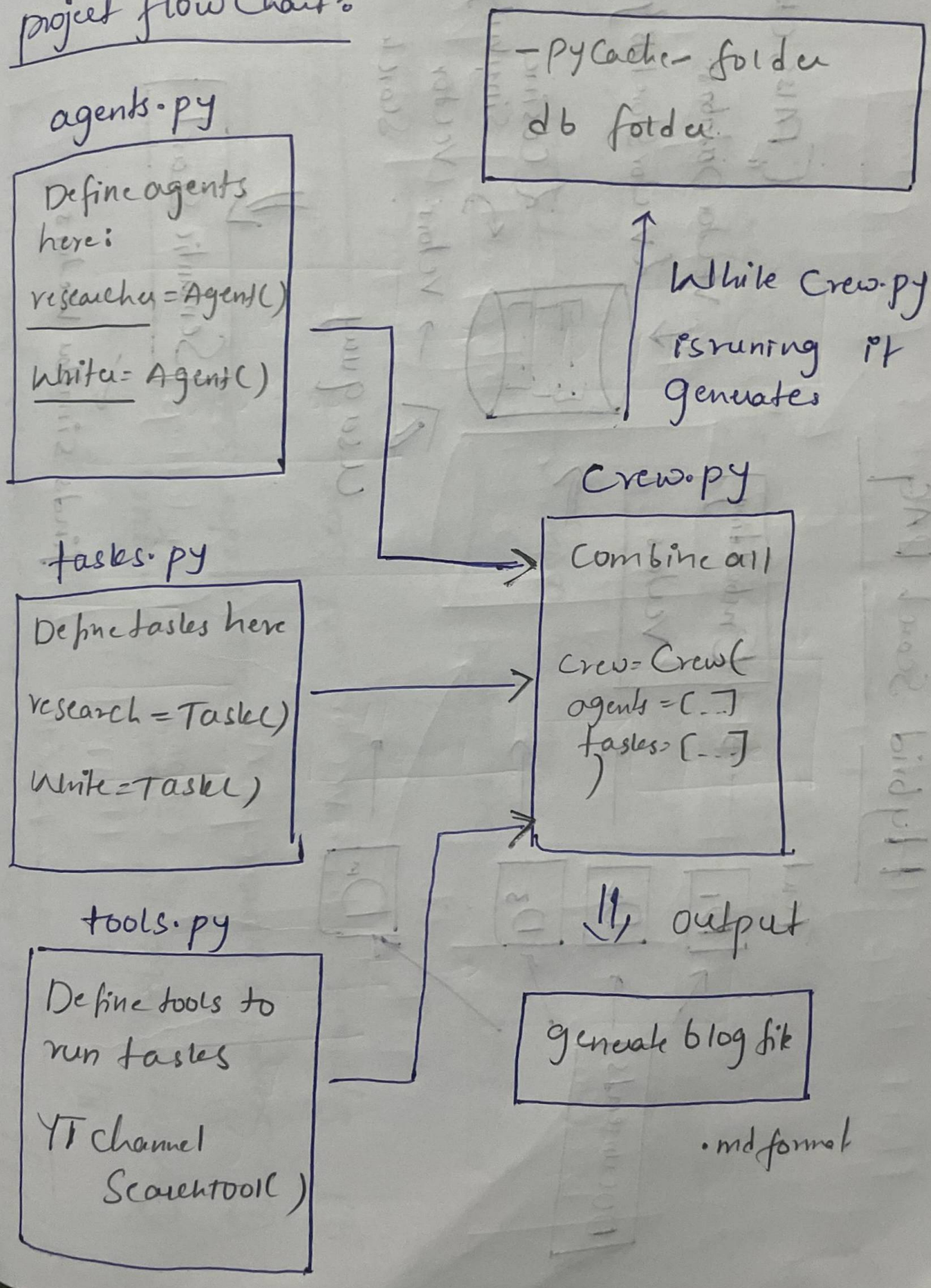
Output

project 14:

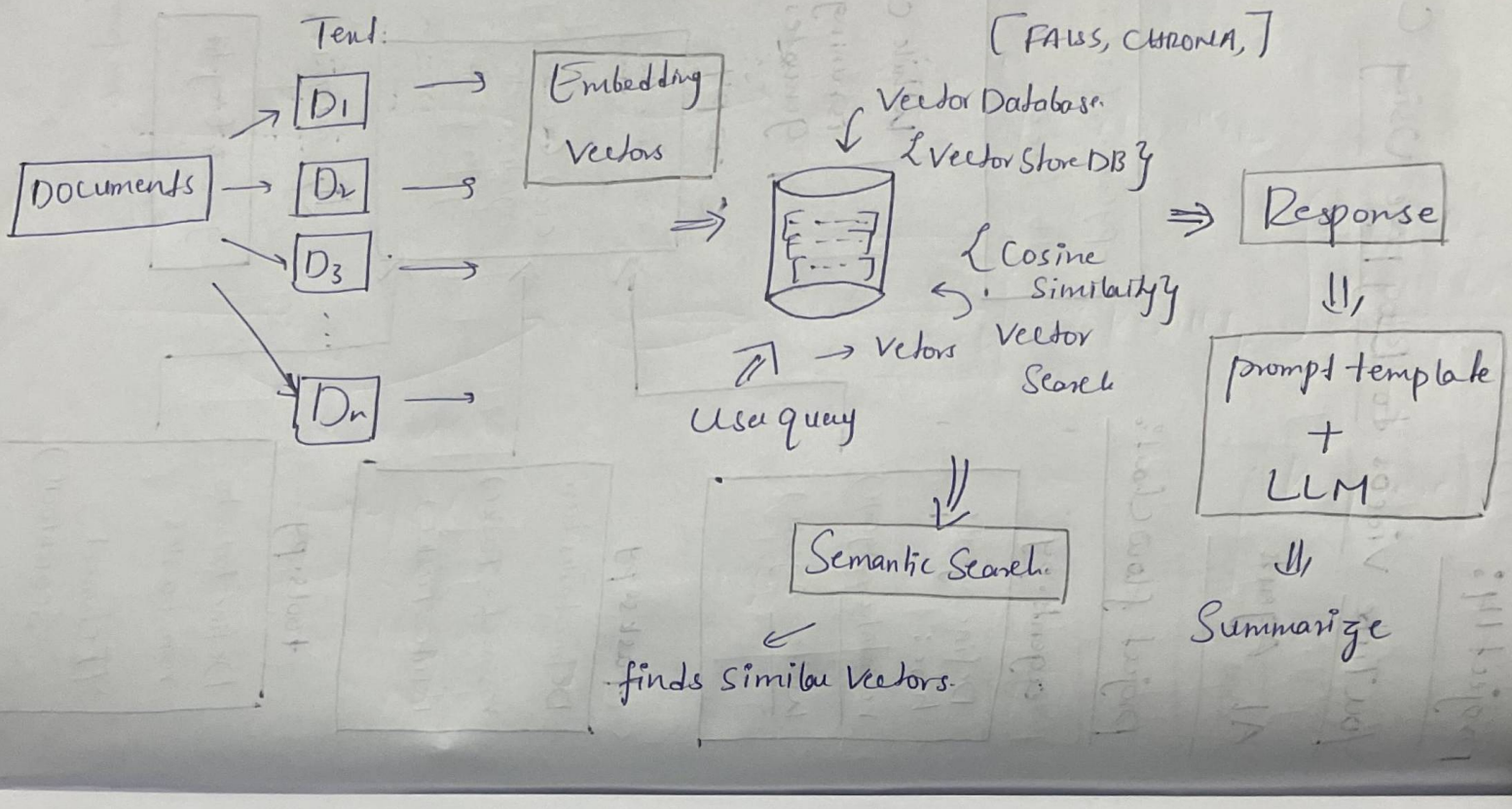
YouTube Videos to Blog page Using CREW

AI Agents

project flow chart:



Hybrid Search RAG



Hybrid Search Mechanism: here we are not just focusing on Vector dense Search.

⇓

Combines Multiple Search techniques.

① Semantic Search → Dense vector Search → Similar.

② Syntatic Search → Exact Search → keyword Search

↳ Sparse Vector Search;

⇓

text → Sparse matrix.

↳ By using OHE, BOW, TF-IDF

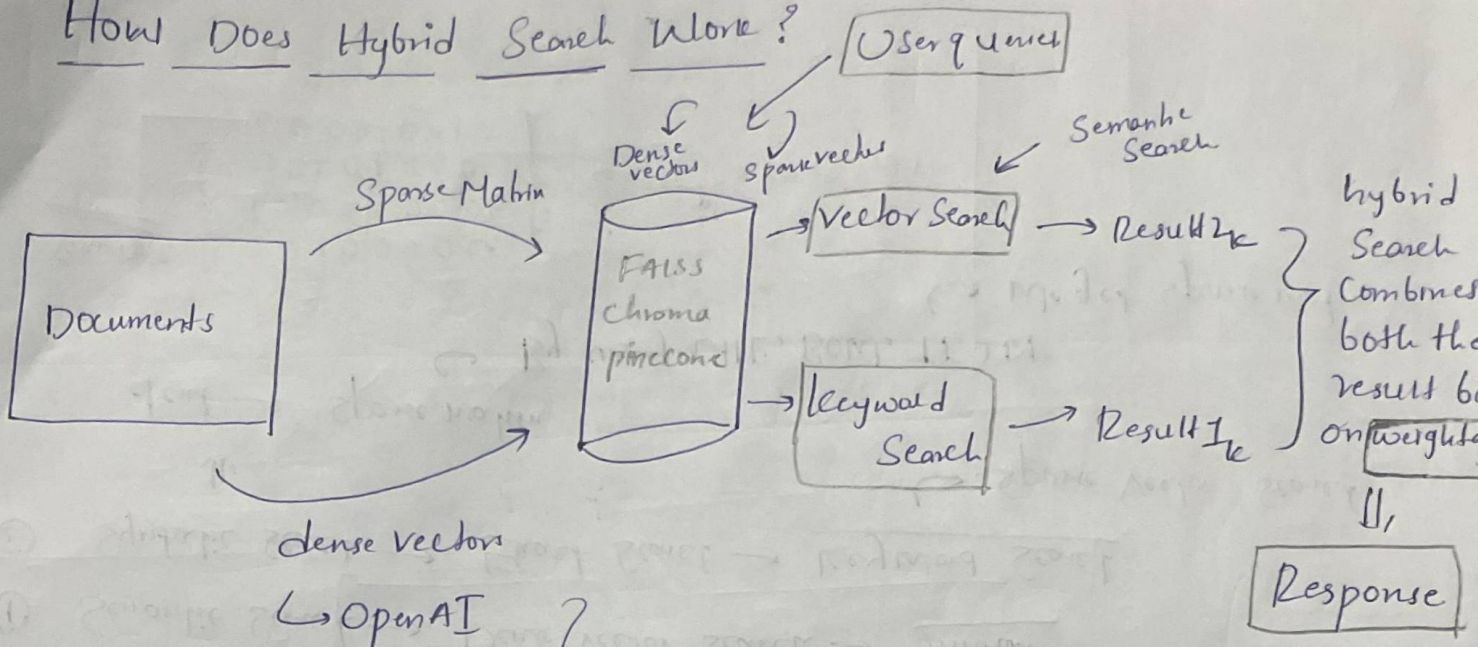
↳ We get Sparse matrix.

Example

$$\begin{bmatrix} 0\,0\,1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,1 \end{bmatrix}$$

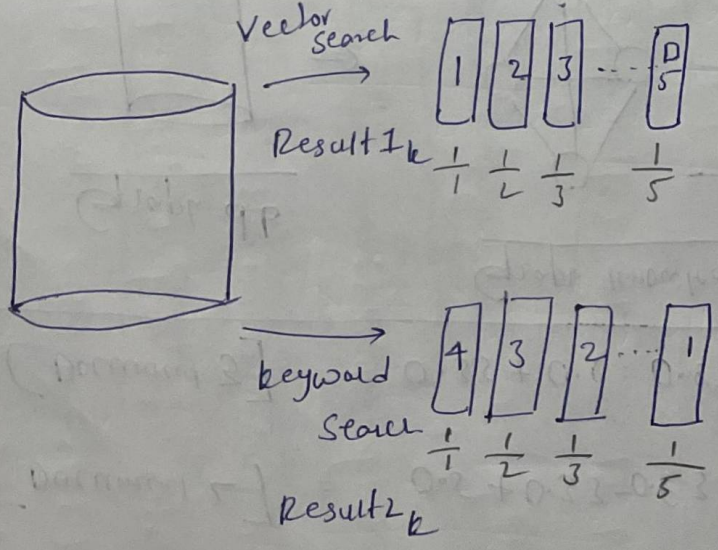
⇐ Sparse matrix

How Does Hybrid Search Work?



How Weightage is given : Based on the technique

Reciprocal Rank Fusion in Hybrid Search



$$\text{final score} = \sum \frac{1}{(c + R_d)}$$

$$= \sum \frac{1}{R_d}$$

$R_d \rightarrow$ Rank of documents.

$c \rightarrow$ constant

depends on various databases

1 → 60

[Document 1] = 1 + 0.2 = 1.2

[Document 2] = 0.5 + 0.33 = 0.83

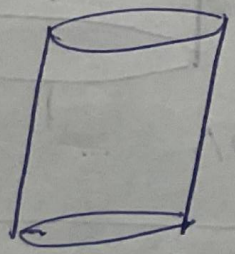
[Document 3] = 0.33 + 0.5 = 0.83

50% ⇒ keyword Search

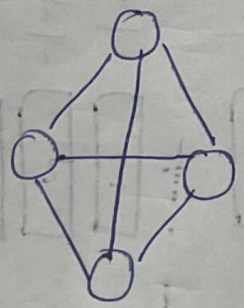
Semantic Search.

Graph knowledge Search

Graph db



⇒



→ keyword Search

→ Semantic Search

→ Graph knowledge Search

use query

Graph db {Neo4j}

Graph DB with LangChain

1. Knowledge Graphs
2. Neo4j Graph Databases
   - ↳ Cypher Queries
3. RAG application with Graph Databases

LangChain

↓ amazing module

langgraph

↓

Where we can create

Multi AI agents

↓ particularly uses

knowledge Graphs

① Knowledge Graph:

also known as Semantic Network

↓

Network of real world Entities

Eg: Events, situation, Concepts

↓

illustrate the relationip blw them

NLP Use Case:

"Rohit Sharma is the Captain of India"

entity entity

Knowledge Graphs:

Three main Components

① nodes → any object, place, person (or) Noun

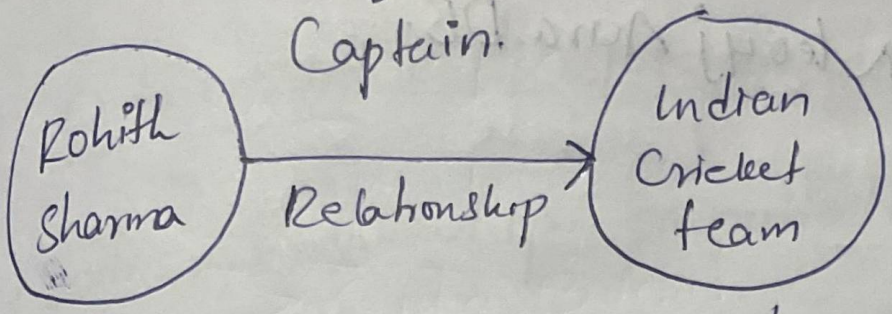
② Edges → define relationship blw nodes

③ Labels

Label → Specifies relationship

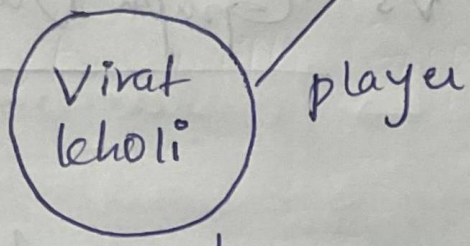
↓

Captain.



node

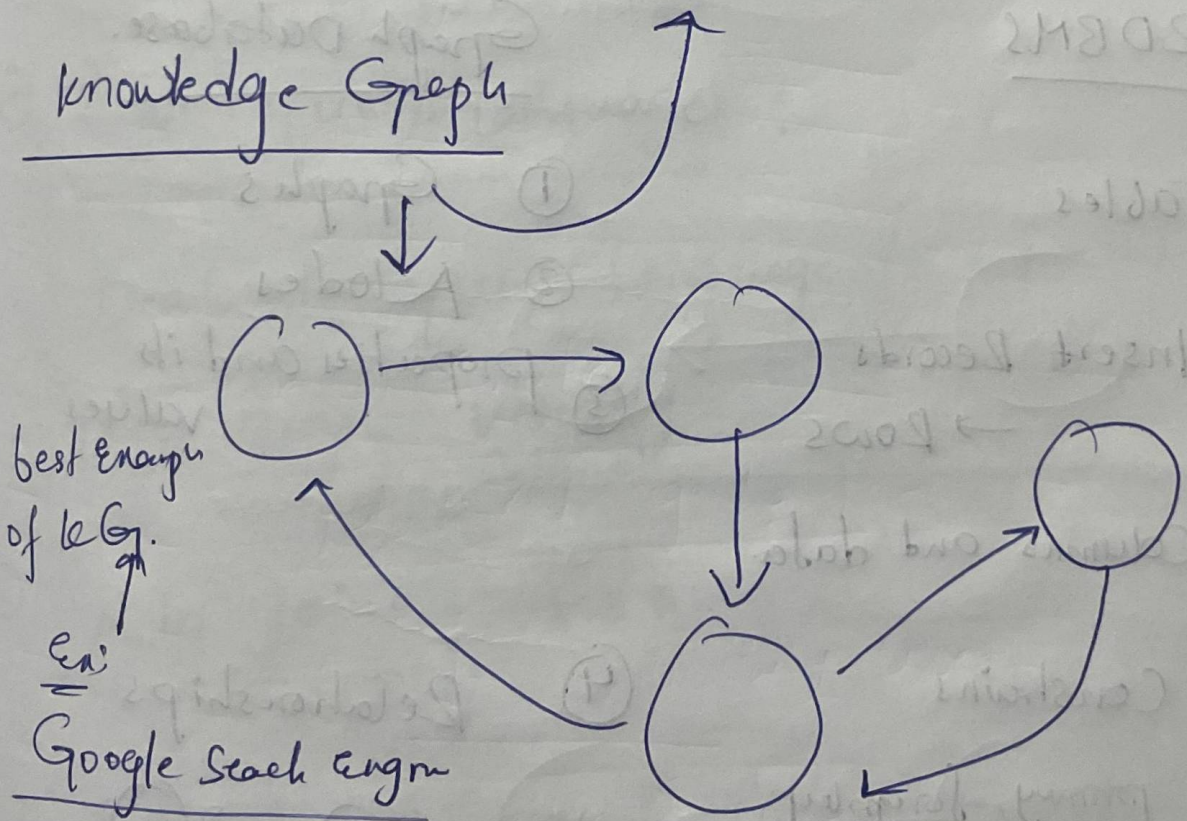
node



node.

"Viratkholi is the player of Indian Cricket team"

Knowledge Graph



best example of KG.

Ex:

Google Seach engine

Search: virat kholi

↓

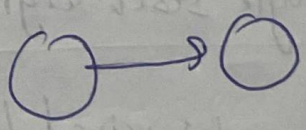
Suggestion → Other players

→ Create Neo4j database from Neo4j Aura DB.

RDBMS vs Graph Databases

↓

Relational DBMS → MYSQL, postgres, Sql/Serva --

RDBMS	Graph Database
① Tables	① Graphs
② Insert Records → Rows	② Nodes
③ Columns and data.	③ properties and its values
④ Constrains primary, foreignkey etc...	④ Relationships 
⑤ Joins Queries	⑤ Traversal.

Advantages of Neo4j

① **Data model:** Responsible for creating Nodes, Relationship, properties and values

② **Real time insights:**

③ **Easy Retrival:** Cypher query like SQL query

④ **Cypher query language:**

Declarative query language to represent the graph visually.

⑤ **No joins** ☆☆☆

⑥ **Supports ACID property:**

atomicity, Consistency, Isolation, Durability

⑦ No fixed schema, instead supports flexible schema.

Neo4j property Graph Data model

Neo4j Graph Database

↓

has property Graph Model

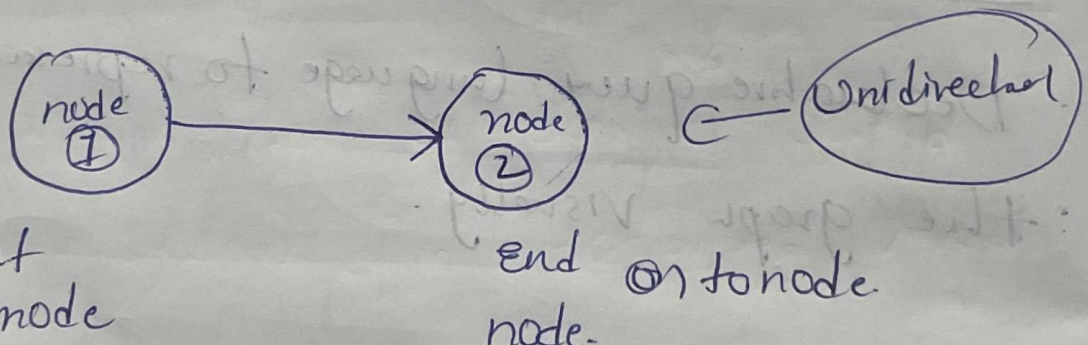
⇓ ↳ to store and manage data.

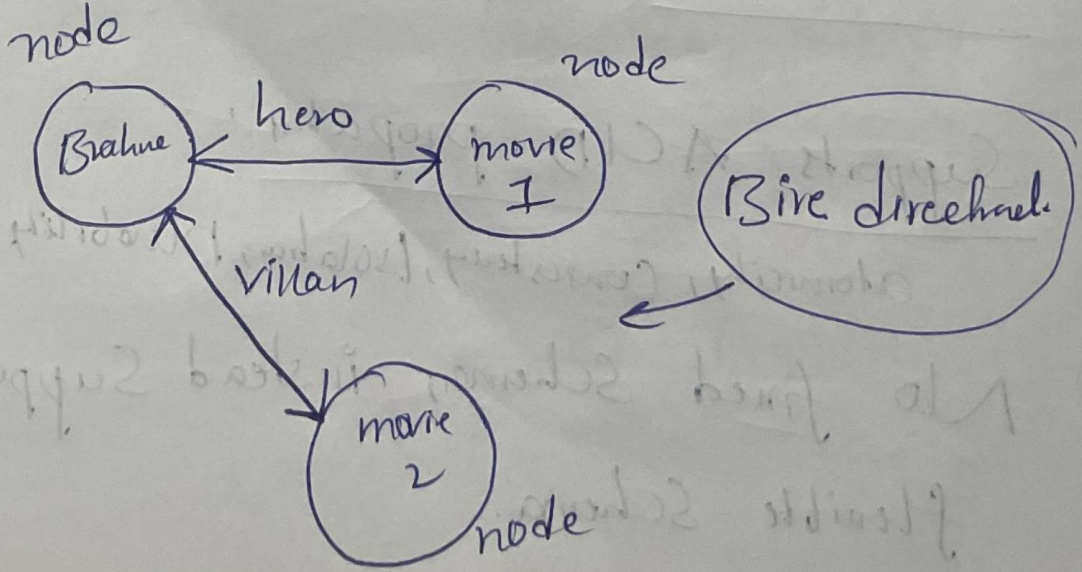
Data = { Nodes, Relationships, properties }

⇓ ⇓

Unidirectional, Bidirectional

key, value pairs





Getting Started with Cypher Query language

① Create(Model)

② Create (Elon : CEO)

node : value

③ Create (Elon : CEO : Employee)

④ Create (Elon : CEO { name : "Elon musk",

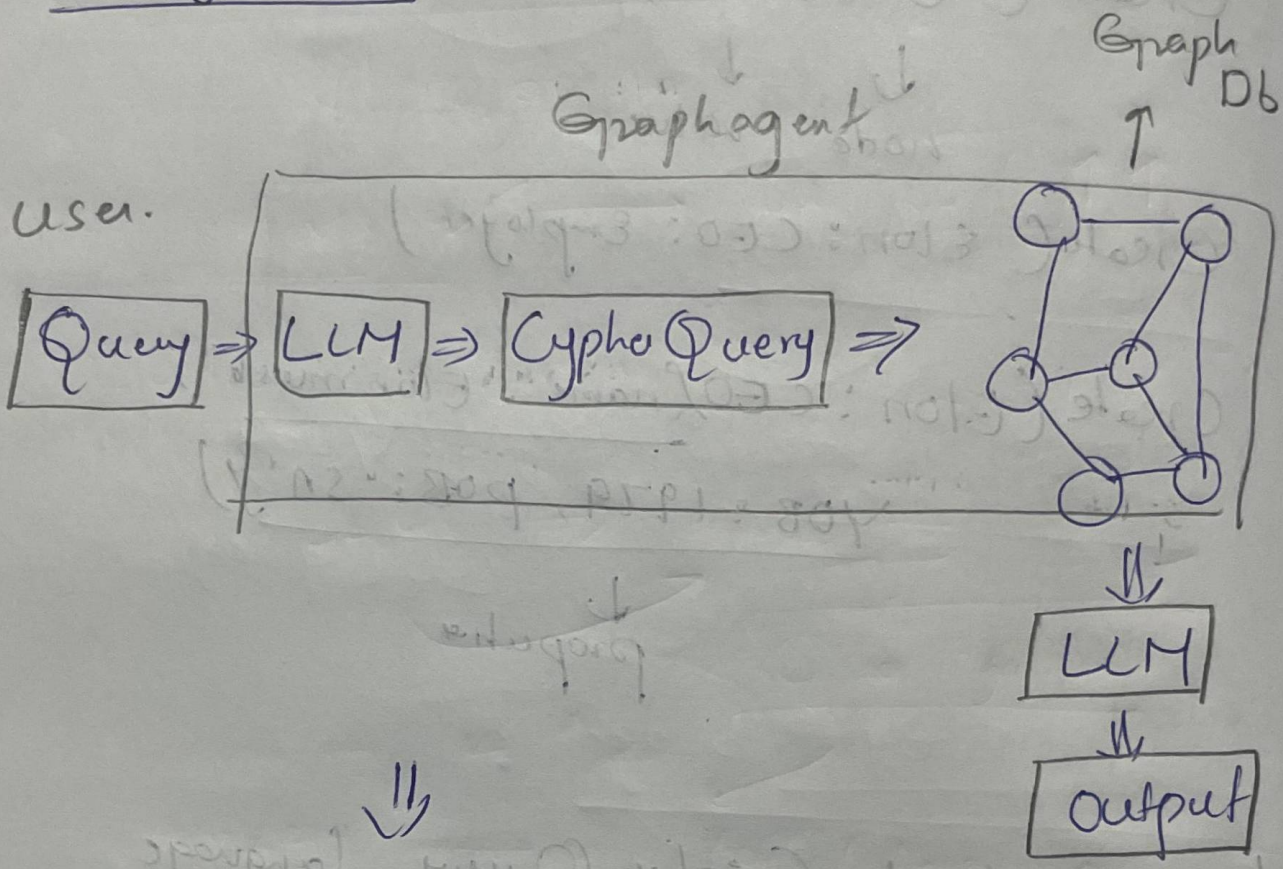
YOB : 1979, POB : "SA" })

property

Learn about Cipher Query language that really helps you to create your graph data bases.

Project 15:

Practical Implementation of Graph Db with LangChain :



Tried my best, but does not worked in my system, due to changed Version and Compatibility issues.

What is Quantization Indepth intution

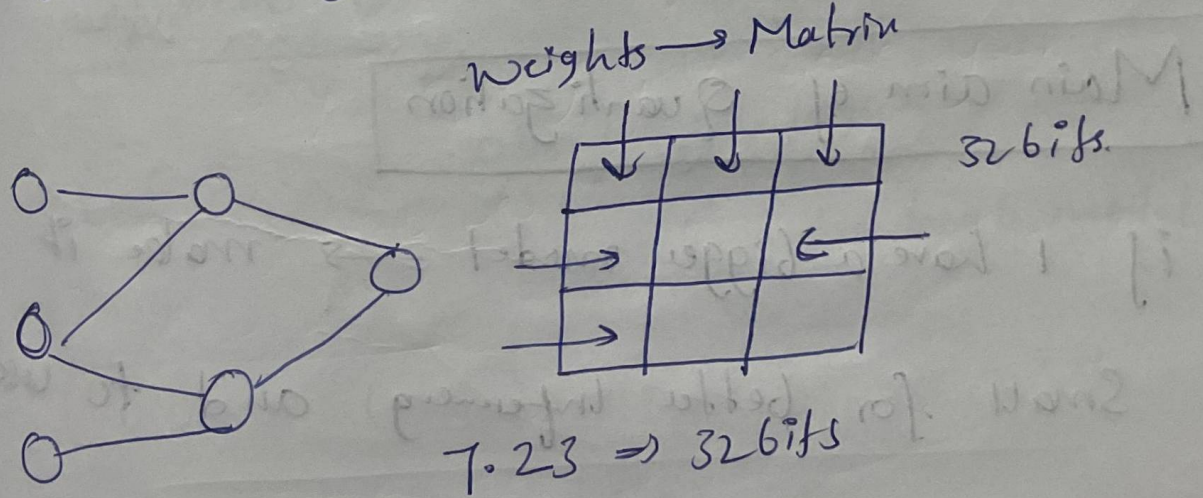
Quantization : → Data → weights & parameters.

1. Full precision / half precision
2. Calibration → problems
3. Modes of Quantization

- post training quantization
- Quantization aware Training.

Quantization :

Conversation from higher memory format to a lower memory format.



FP → Full precision / single precision

floating point 32 bit

if you want to load

Llama 2 70b it is not supported by your System with 32gb Ram.

↓ to do that

Aws ✓ ⟶ but cost ↑↑

So that we convert

32bit ⇒ int 8

⇒ Quantization

Very important for Inferencing

Main aim of Quantization.

if I have a bigger model ⟶ make it small for better Inferencing and to use

- in mobile phones, edge devices.
- Smart watches etc..

We can also fine tune the model after quantization but there is loss of accuracy

↓

due to loss of memory

↓

there are techniques to overcome it

How to perform Quantization.

① Symmetric Quantization ⇒ Batch normalization.

② Assymetric quantization.

① Symmetric Uint8 quantization

[0.0, ... 1000] → weights ×

↳ for larger model.

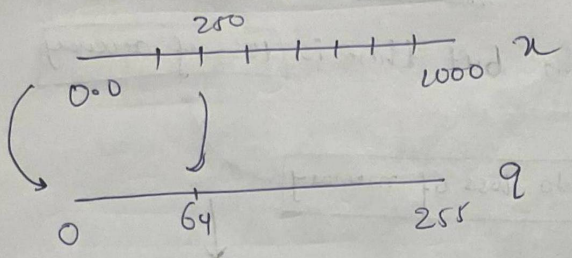
[0.0 ... 1000] → number → 32 bits

↓ Quantize

Uint 8 ⇒ 8 bit ⇒ 2^8

→ 0–255

↙ Symmetric data distribution



Minimum Value

0.0 → 0

1000 → 255

$$\text{Scale} = \frac{x_{max} - x_{min}}{q_{max} - q_{min}}$$

$$= \frac{1000 - 0}{255 - 0} = \boxed{3.92}$$

↓ Scale factor

$$\text{round}\left(\frac{x}{\text{Scale factor}}\right) = \frac{250}{3.92} = 64$$

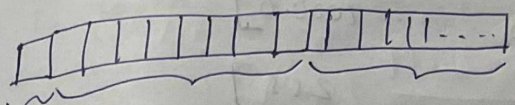
↑ Scale factor

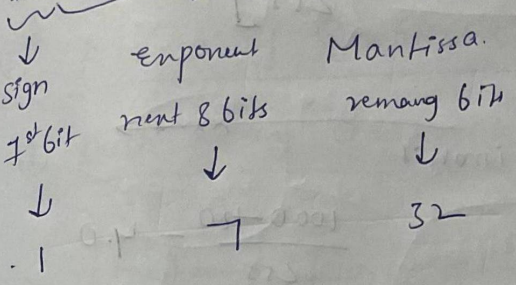
← quantized value

Symmetric quantization

Single precision Floating point 32

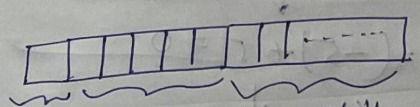
How number is stored

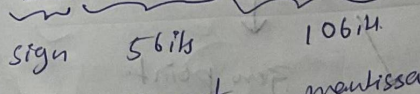




+7.32

For FP 16





(ii) Assymetric quantization:

Assymmetric Uint 8 quantization

[−20.0, ... 1000.0]

[0, ... 255]

−5.0

↓

but how can i make it to zero

$$\frac{1000+20}{255} = 4.0$$

↓

Scale factor

Convert −20

$$\text{round}\left(\frac{-20}{4}\right) = (-5) + 5 = 0$$

↓

zero point

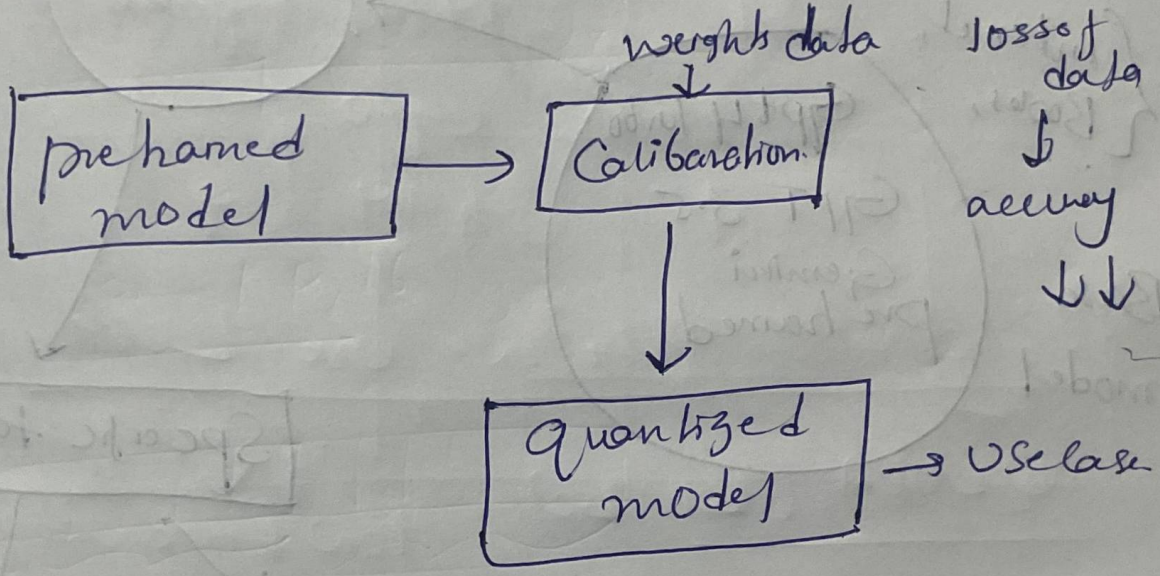
Which is not on scale that why we are adding zero point of same number.

② Calibaration:

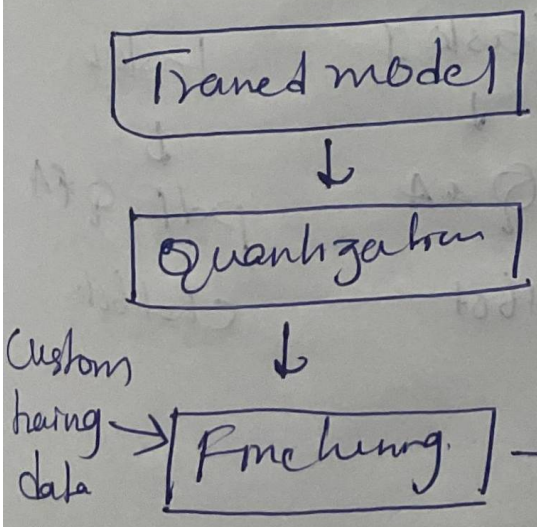
Squeezing the values from higher to lower is called Calibaration.

③ Modes of Quantization.

① post training quantization. (PTQ)

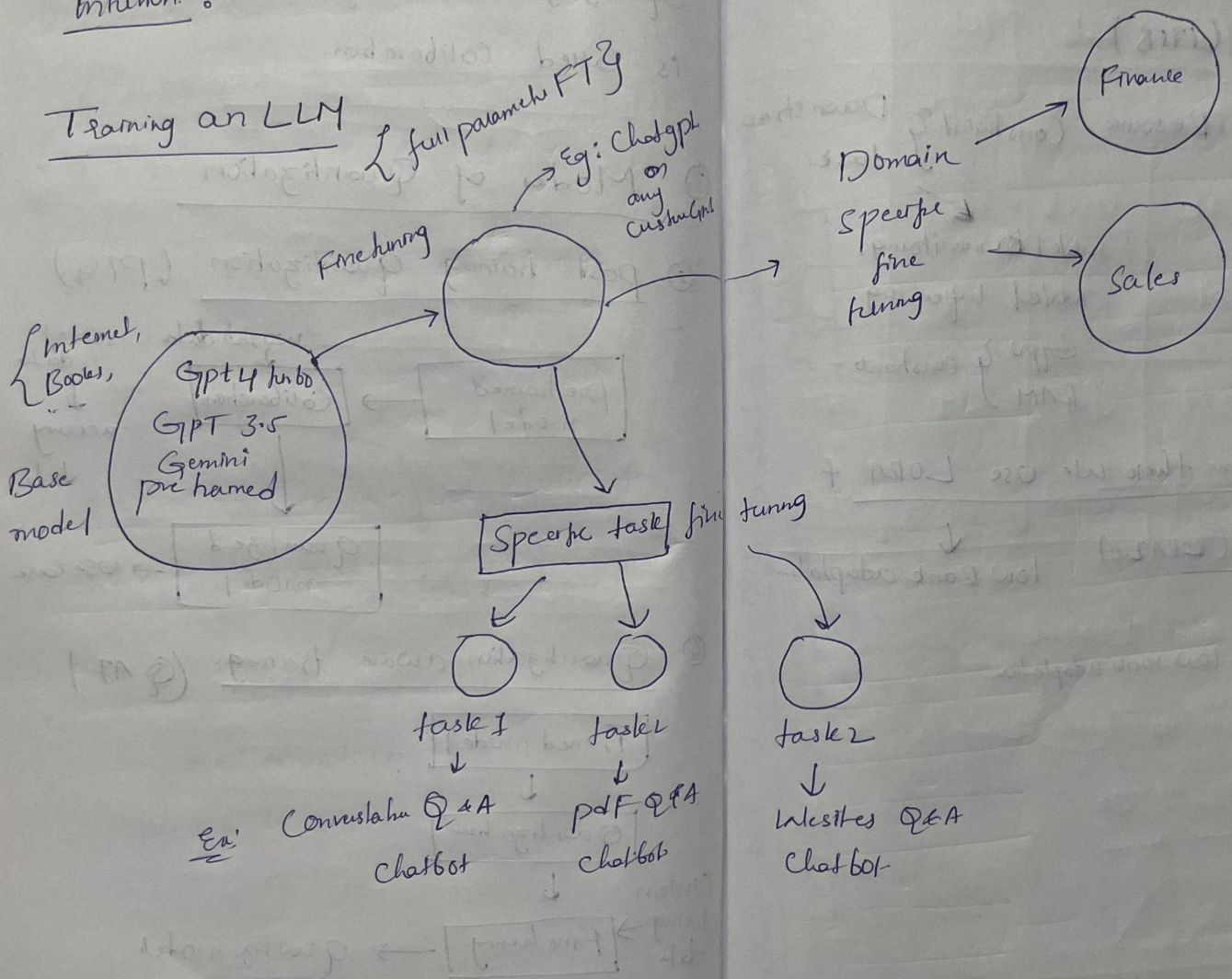


② Quantization aware training. (QAT)



LORA & QLORA Indeagh mathematical intuhon :

Training an LLM



Full parameter Fine Tuning [challenges]

① update all model weights (175B)

② hardware Resource Constraint } Downstream task

↓

model monitoring,
model inferencing
GPU
RAM } constraints

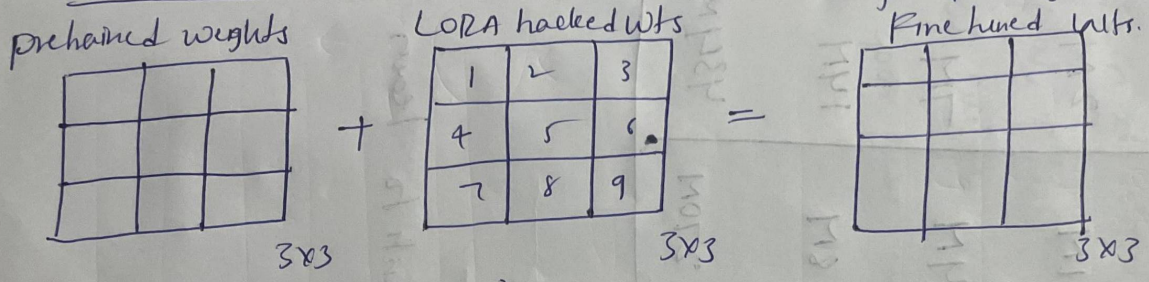
To overcome these we use LORA & QLORA [QLORA2.0]

↓ low Rank adaptation

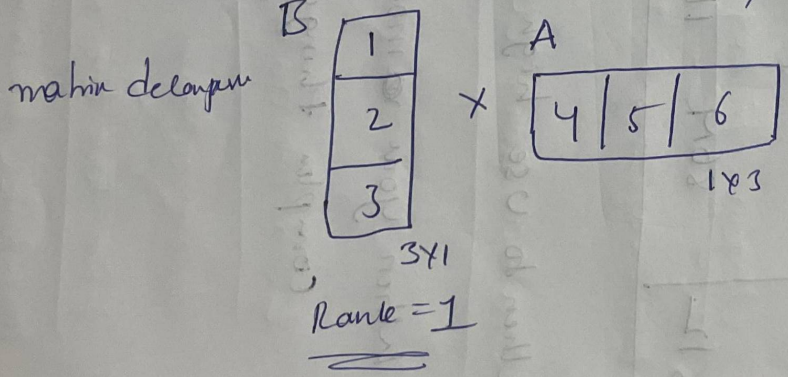
↓

Quantization low rank adaptation

What does LoRA DO? Instead of updating weights, it tracks changes.



$W_0 + \Delta W = W_0 + BA$



Rank = 1

No of trainable parameters

Rank	7B	13B	70B	180B
1	167K	228K	529K	849K
2	334K	456K	1M	2M
8	1M	2M	4M	7M
16	3M	4M	8M	14M
512	86M	117M	270M	434M

When to use high rank

When your model want to learn complex things

End to End FineTuning LLM Models with Lamini platform: Project 16

Lamini platform:

is a developer friendly platform that enables fast, efficient fine tuning and deployment of custom large language models using your own data on any hardware.

Project 16: End to End Fine tuning LLM models with Lamini platform

Steps

① Define the data

- list of dictionaries
- input, output
- this is your training data.

get_data()

② Import lamini library

```
import lamini
```

③ Set your api key

```
lamini.api_key = "- - - - "
```

④ Create a Lamini LLM object:

```
llm = Lamini(model="llama3-8B-Instruct")
```

⑤ Load the data:

```
data = get_data()
```

⑥ Fine tune the model:

```
llm.tune(data_or_dataset=data,
         finetune_args={'learning_rate': 1.0e-4})
```