Step 1

Generative AI

- What is GenAI?
  - type of AI that creates new content, text, images, code
    ⇓
    Generative AI
    ↓
    [mimicking human creativity]
- GenAI Impact Areas:-
  i) Customer Support
  ii) Content Creation (Human creativity)
  iii) Education (ChatgPT) ← Teacher
  iv) Software Development
- Is GenAI successful?
  - Does it solve real world problems?
    Yes
  - Is it useful on a daily basis?
  - Is it creating new jobs. (Yes in AI)
- How To learn in (everyday advancement
  - آپ سنتے ہیں کہ foundation models کو ڈال سکتے ہیں

User perspective like Client — Foundation Models — Builder perspective like Big Company

Prompt Engineering RAG AI Agents Vector Databases Fine Tunning	RLHF pretraining Quantization Fine Tunning

- The Builder's Perspective:-
  i) Transformer Architecture.
  ii) Transformer Types:-
      Encoder Only (BERT)
      Decoder Only (GPT)
      Encoder and Decoder based (T5)
  iii) PreTraining
      Training Objectives
      Tokenization strategies
      Handling Challenges
  iv) Optimization
      Training optimization
      Model compression
      Optimizing Inference
  v) Fine Tunning:
      Task Specific Tuning — RLHF
      Instruction Tuning — PEFT
      Continual Pretraining
  vi) Evaluation
  vii) Deployment

The User's Perspective
(How To build application using LLMs)

i) Building Basic LLM Apps
   Open Source vs Closed Source LLMs
   fetching api using Hugging Face
   Using LLM APIs
   LangChain
   Hugging Face
   Ollama

ii) Prompt Engineering (How To use and get ada

iii) RAG

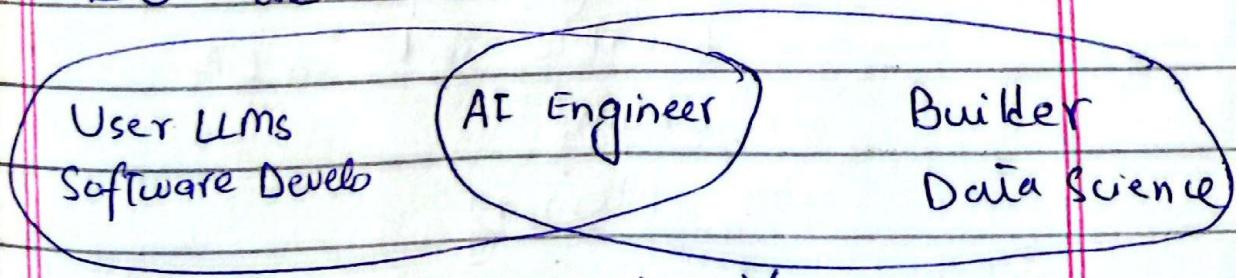
iv) Fine Tuning

v) Agents (like chatgbot but do extra work

vi) LLMOps

vii) Miscellaneous

- Do we need To learn both



The answer is Yes.

LangChain

→ User's perspective

→ first step because it integrate with all other Tool's

LangChain is an open-source framework that helps in Building LLMs based application. It provides end-to-end tools to developers build complex AI applications such as chatbots, question-answering systems, RAGs.
Support all LLMs.
Support all major GenAI usecases.

PlayList Videos

① Fundamental :-
- i) What is LangChain
- ii) Models
- iii) Prompts
- iv) Parsing Output
- v) Runable and LCEL
- vi) Chains
- vii) Memory

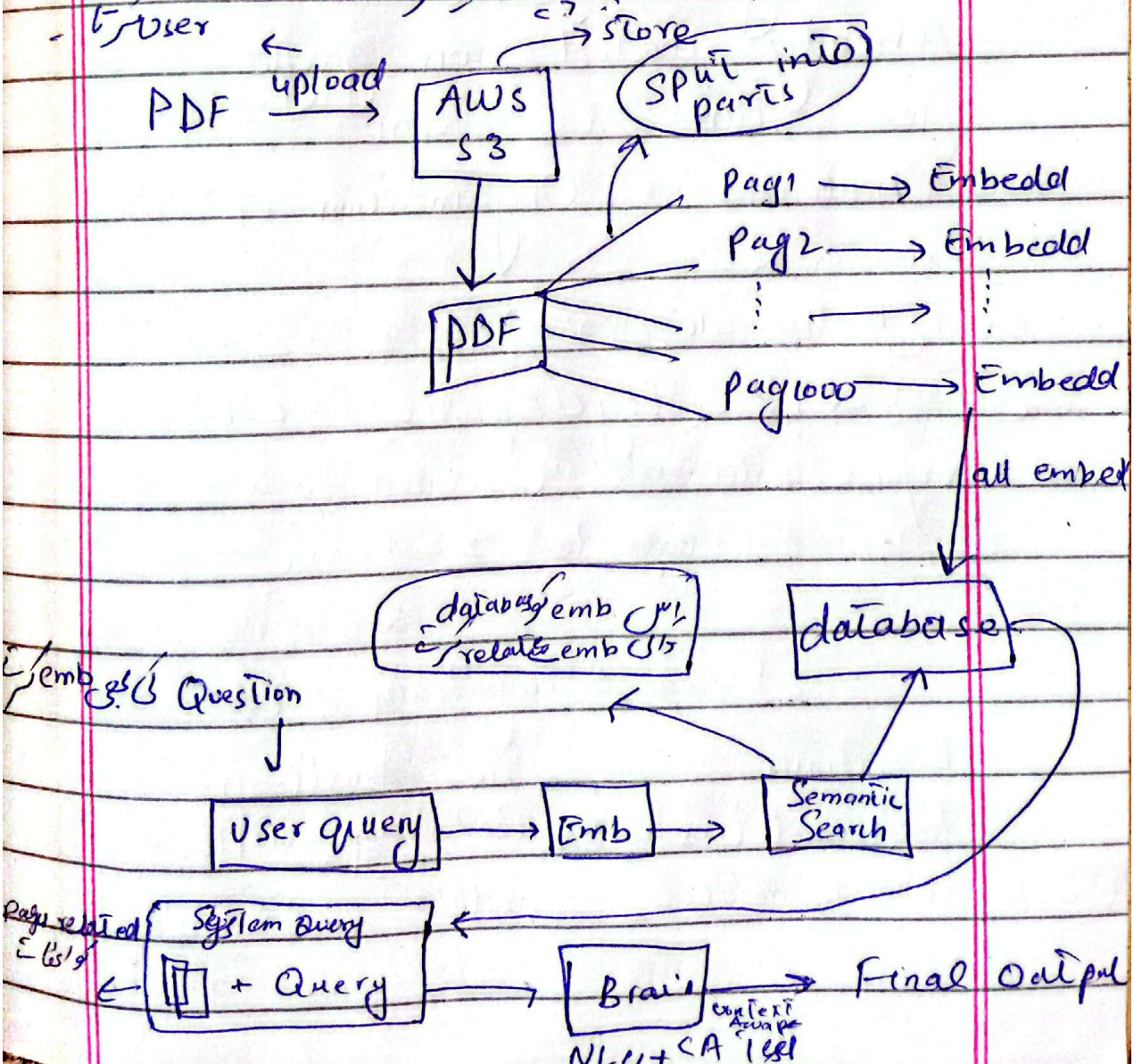
② RAG
- i) Documented Leaders
- ii) Text Splitters
- iii) Embeddings
- iv) Vector Stores
- v) Retrievers

vi) Building a RAG application

③ Agents
i) Tools and Toolkits
ii) Tool Calling
iii) Building an AI Agent

Why do we need langchain.

مثال کے طور پر ہم نے ایک web بنائی ہے جو pdf reader ہے اور اس میں chat option بھی ہے آپ سوال پوچھ سکتے ہیں۔ 2014 میں اگر آپ کو ایسی web بنانی ہوتی ہے تو آپ سے کیسے کرتے۔



- Challenge 1 - Building Brain

Query understanding + relevant text semantic

So we just add LLM's which have these capacity.

- Challenge 2 - LLM'S Weigh

It is very heavy. On loading Server it take many computation. So many companies faces these solution and get solution.

Now you can not load on server. Already loaded on google server you can use api.

- Challenge 3 - Organization of these components.

اگر آپ نے یہ سارا سسٹم ہاتھ سے code کیا اور کل آپ کو پتہ چلتا ہے کہ آپ کی OpenAI کی Key ختم ہو گئی ہے یا دوسرا ڈاؤن زیادہ perform کرے model اور کوئی embed میں change ہو گیا ہے تو آپ کو دوبارہ سے یہ code لکھنا ہوگا اور پھر سے code لکھنا ہوگا۔

یہیں پہ Langchain کا تصور آتا ہے۔

Langchain has the built-in functionalities. تو ان سارے Components کو اکٹھا کرتا ہے۔ اور ایک لائن کے code سے آپ model change ہو جاتا ہے۔

Benefits

i) Concepts of Chains.

ہمارے پاس پر Task کے بہت سارے Compon ہوتی ہیں
اور جب ہم انہیں pipline میں ڈالتے ہیں تو ہم ایک
chain بنا سکتے ہیں جس میں ہر قسم perform Task کو
کر سکتے ہیں۔ اور پھر ہر Task کا output دوسرے
Task کا input بن سکتا ہے۔

ii) Model Augunasted Development.

آپ اپنے ہر Task میں کوئی بھی model کا استعمال
کرو۔ اس سے فرق نہیں پڑتا ہے۔ آپ جب
سے change model کر سکتے ہو۔

iii) Ecosystem Compluted:-

langchain نے کوشش کی ہے مکمل ecosystem دینے کی
Doc loader میں کوئی بھی document load کر سکتے۔ excel, pdf
embedding میں بھی بہت سارے avail models کو بھی
database میں بہت سارے ہیں۔

(iv) Memory and state handling:-

اگر ایک conversation میں Linear Reg کی بات
ہو رہی ہے اور اگر کوئی conversation میں LR کو بغیر mention کے
اس کے related کچھ بتا سکے گا۔

- What can you Build?

i) Conversation Chatbots

chat bot —like→ call center

ii) AI knowledge Assistent

also Trained on your data.

(iii) AI Agents

also give some suggest also do work for you

book your tickets like a person or agent talk like human work like human.

(iv) Work flow automation

(v) Summarization & Research helper.

- Alternative

Llammaidex. ai

Haystack.

Video #2 Components of LangChain

1. Models.
2. Prompts.
3. Chains
4. Memory
5. Index.
6. Agents.

1. Models

are core interface which interact AI.

NLP task => ChatBot

Challenge1 : Natural Language Understanding

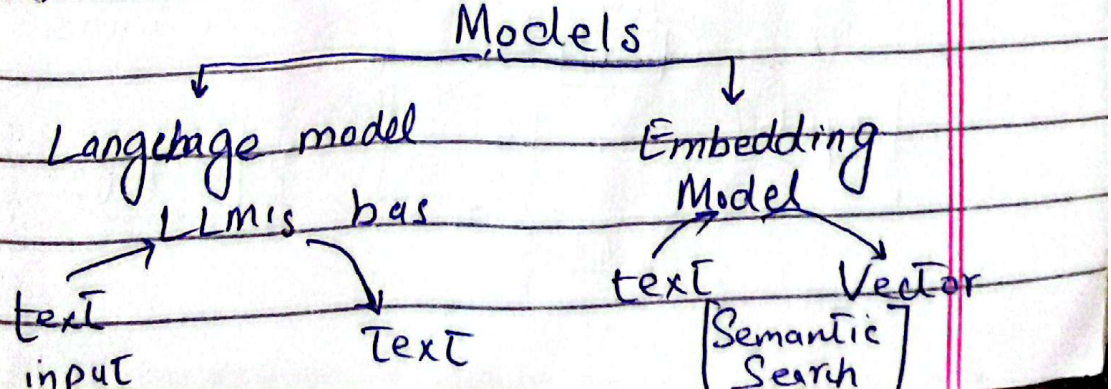
Challenge2: Reply => Context aware text generati

Problem & Challenges solved by LLMs

Challeng 1:- Huge size of LLM's which small comany drive on server

Solution:- Builder company make API around LLM's.

Challenge 1:- Implementation:- Langchain make interface that everyone comunate with every company ai model



2. Prompts:-

LLM → input = prompt

Chatgpt: What is langchain, = prompt

Prompt is very very important

So prompt Engineering comes

Types:- i) Dynamic and Reusable {Summarize Topic}

ii) Role_based

Hi you are experienced {profession}

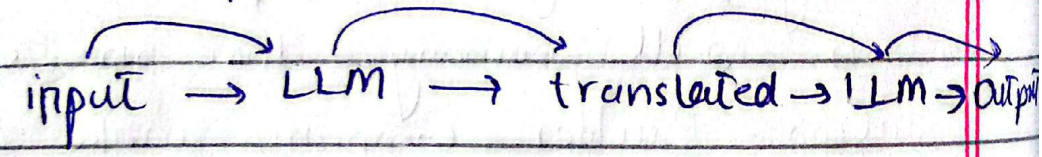
Tell me about {topic}.

iii) Few Shot

3. Chains:-

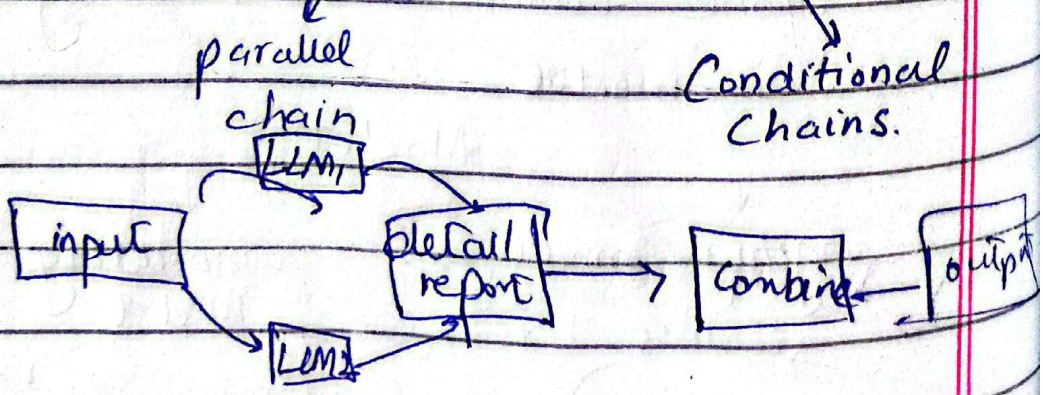
To build pipeline

LLM application → pipeline



Use chain which do one input others output

Complex pipeline



4) Indexes:-

Indexes connect your application to external knowledge — such as PDFs, website or databases.

Components of Indexes:-

- Document Loader
- Text Splitter
- Vector storage (Database)
- Reterivals

Chatgpt has no excess to external sources of your company So, it cannot answer you about you company privacy.

So we build LLm's and external knowledge to these LLm's, so it trained on these data also.

" Indexes is the way to build LLM based application which has excess to external knowledge source."

(5) Memory

LLM's API calls are statelless

stateless کا مطلب ہوتا ہے جب آپ LLM API پر call مارتے ہیں۔ تو ہر independent request ہوتی ہے یعنی اسے previous call کے بارے میں کوئی understanding نہیں ہوتی۔

compon :- Langchain کے کون سے problem solve کرتا ہے؟
This is the main problem. . Model.

Q1:- prompt → [Who is Narendra Modi] → [LLM API] →

[Narandra Modi is the Indian Prime Minister Since May 2014]

Q2: [How old is he?] → [LLM API] →

As an AI, I don't have access to personal data about individuals unless it has been shared with me.

i) Conversation Buffer Memory:-
پچھلی تمام باتیں جیت چیت کو store کرتی ہے
اور next API call کرتے وقت بھیج دیتی ہے۔

ii) Conversation Buffer Window Memory.
store کرتی ہے interaction lastN سے
اور call API کرتے وقت بھیجتی ہے۔

iii) Summarize-Based Memory:-
Summary of older chat.

iv) Custom Memory:-

AI Agents

LLM property → NLU → Text generation

Chatbot

Chatbot :- صرف سوال کا جواب دیتے ہیں۔

AI Agents :- آپ کے لئے کام بھی کرتے ہیں۔

(Chatbot with super power)

When you ask question, AI agents hit on API and do answer.

AI Agents [Reasoning capability
Tools Access (API hit)

Tools => Calculater, Weather API

~~to~~ Can you multiply 3 with tempreture of Lahore?

First he chech an API who give him tempreture.

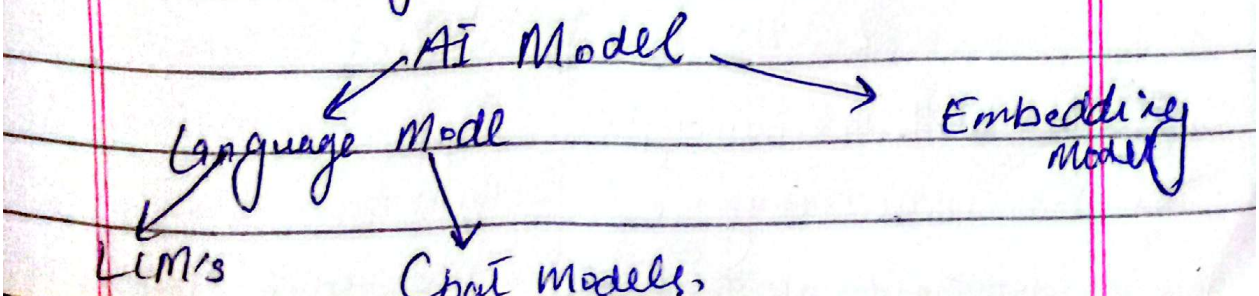
then he search calculator and he call calculator.

= 25°C * 3 = 75

Video # 3

Model Component

"Model component in langchain is nothing but an interface which helps you connect with different kinds of AI Model.



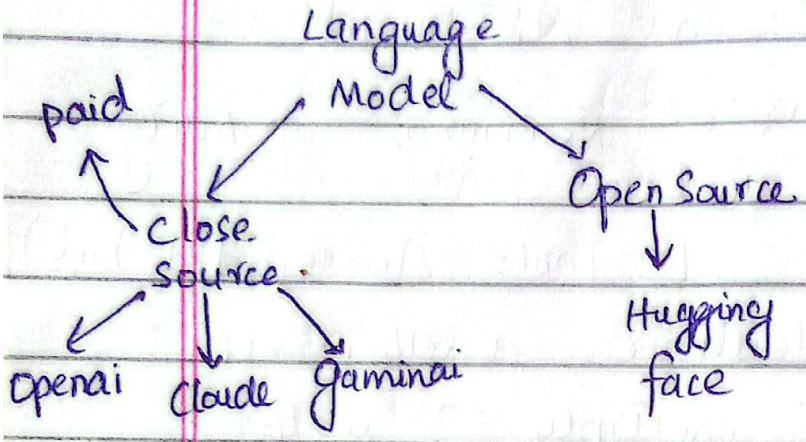
Language Model ↑ Chatbot ↑

Text → AI Model → Text

Text → Embedding → [.....] vector

for Semantic Search

Rag Based Application



Language Model.

Types :-

i) LLM's

ii) Chat Model

LLM's :- General purpose model that used in raw-text-generation. There are older model.

Chat Model :- Language Model that are specialized for conversational task. They take a sequence of message as input and return chat messages as output

LLM's	Chat Models
No built in memory	Support structured
No understanding of user and assistant role	Understanding system user and assistant

Use Cases:- (LLM's)

- Text generation, summarization, translation creative writing, code generation
- Conversational AI, chatbots, virtual Assistent. customer support, AI tutor.

Setup:-

LLM's Models

i) Openai
ii) Anthropic
iii) Google
} closed_source, paid

iv) Hugging face ⟶ Opensource

Openai_Langchain

```
from langchain-openai import OpenAI
from dotenv import load_dotenv
```

* dotenv load your api key from .env file

```
load_dotenv()
llm = OpenAI(model="gpt-3.5-turbo-instruct", temperature=(0-2) => range, max_completion_tokens=10)
result = llm.invoke("This is your prompt")
```

max_completion_tokens = for words

#invoke is everywhere in langchain for every ai model To give prompt of ai model and get answer

```
print(result).
```

" string input ⟶ string output (llm)

Chat Models

Same code as Llm code.

- from langchain_openai import ChatOpenAI
- from dotenv import load_dotenv

```
load_dotenv()
model = ChatOpenAI(model='gpt-4')
result = model.invoke("Something")
print(result)
```

Anthropic and Chat Google generative ai is same as code this openai

Open Source AI Model

are freely available AI model than be downloaded, modified, fine-tuned and deployed without restriction.

"Closed Source Mode is on the server on builder company. These company gives API to user and get back many. These model running on server"

"Closed Source Mode is on the internet not on company server. It is

uploaded on internet. you can just download them.

" You have control on it "

" You have data privacy of your company. "

Where To find?

On Huggingface.

Largest repository of open source LLM's Models.

Open-Source Models

Using HF Interface API ← → Running locally

↓

API like OpenAI but free. you can use thousand of model.

- Disadvantage:-

High Hardware Required

Step Complexity

Lack of RLHF

Limited Multimodel Abilities

Code is same as other.

```
llm = HuggingFaceEndpoint(rapid-id ~~model~~ =
'tinyLlama/TinyLlama-1.1B',
task = 'text-generation'.
model = ChatHuggingFace(llm=llm)
result = model.invoke(...)
```

Models:

Locally HuggingFace downloaded

```
import HuggingFacePipeline
llm = HuggingFacePipeline(
    model_id = '          ',
    task = 'text-generation',
    pipeline_kwargs = dict(
        tempreture = 1.5,
        max_new_tokens = 100
    )
)
```

0 = same
0.1 = some changes
0.7 = more
1.5 = completely changes!

...

Embedding Models.

OpenAI Embedded — OpenSource

```
from ... openai import OpenAIEmbed
from dotenv
load_dotenv()
model = OpenAIEmbedding(model='text-3-large',
                        dimensions=84)
result = model.embed_query('
    What is CampusX')
print(result)
```

→ 1536 - for small embedded model

3072 - for large model.

you can also do multiline query

```
document = ['  ....  ',
            '  ......  ',
            '  ........  ']
result = embedding.embed_document(document)
```

Video # 4: What are Prompts:-

Prompts are input instruction give To model To guide its output.

Prompts → Text-based
Prompts → Multimodel (image, pdf, sound)

- Static prompts.

```
from langchain-opai import ChatOpenAI
from dotenv import load_dotenv
import streamlit as st
load_dotenv()
st.title("CHATBOT HELPER")
user_input = st.text_input("Enter your prompt")
model = ChatOpenAI(model='gpt-4')
if st.button("Summarize"):
    result = model.invoke(user_input)
    st.write(result.content)
```

// user input کے بجائے prompt static بھی دے سکتے ہیں۔ تو اس میں ہم user سے کچھ نہیں پوچھیں گے اور اس سے آپ کا result مختلف ہو جائے گا۔

سے web paper research کے بارے میں ہے اور آپ سے
آپ سے research paper کا پتہ نہیں ہے۔ یا نام یاد نہیں
ہے تو وہ اگر غلط نام لکھتا ہے تو llm کو کچھ اور
result دے دے گا۔ تو اس کے لیے آپ کیا کرتے ہیں
تو آپ کو ایک template بنانا چاہیے۔

Dynamic Prompt

"Please summarize the research paper titled "{paper-input} with the following specification:-
Explanation-style : {style-input}

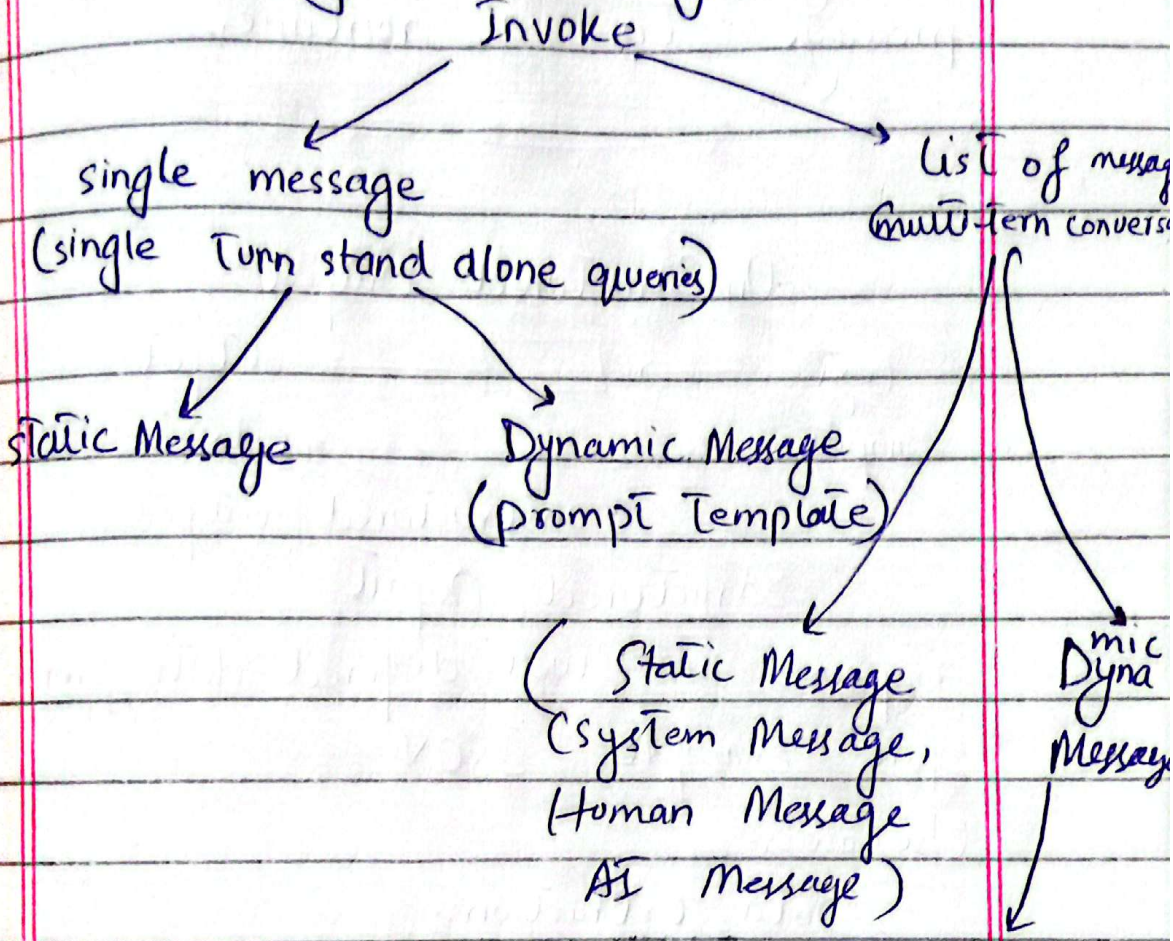
یہ چیزیں ہم user سے input میں ←
لے اور اس prompt میں ڈالے گے اور پھر یہ prompt
ہم llm کو دیں گے۔
سو آپ نے ایک prompt لکھا ہے لیکن اس کی
key values میں وہ user آپ کو دے گا۔

Prompt Template

A prompt template in LangChain in a structured way to create a prompt dynamically by inserting variable into a pre-defined template. Instead of hardcording template allow you to define placeholders that can be filled in runtime in different path.

Why use prompt template over f-string

i) Default validation
ii) Reusable
iii) LangChain Ecosystem.



PromptTemplate :-

PromptTemplate is used for single query.

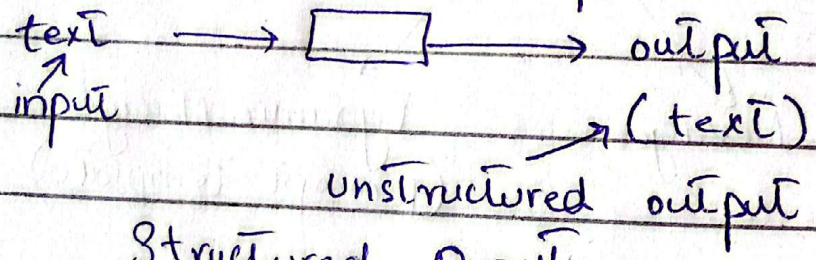
ChatPromptTemplate.

While ChatPrompt Template is used for multiple query using messages.

Message Placeholder:-

in langchain used inside a chatprompt template to dynamically insert chat-history or a list of message to at runtime.

Un Structured Output



Structured Oupt

response in well-defined data format fore example JSON.

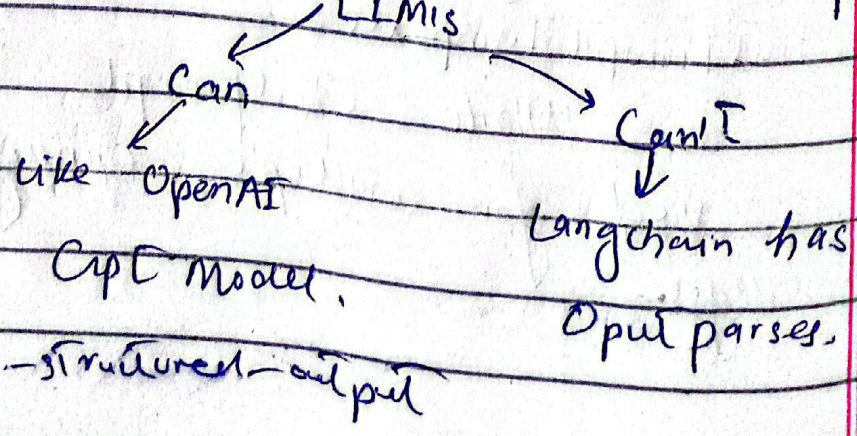
Usecases:-

Data Extraction:-

API Building :-

Agents:-

Ways To get Structured Output



mistral ai/Mistral-7B-Instruct-v0.2
meta-llma/Meta-Llama-3-8B-Instruct

with_structured_output parameter
model invoke k waqt hum
ye duty hain our sath main
data format
TypedDict Pydantic Json

Typed Dict. → Simple
is a way To defined dict in python.

```
Class Person(TDict)
    nam: str
    age: int
```

what the type of the key.

Annotated:-
is the Type of dict which is used To give description To the Typeddict parameter.

```
Class Person(TypedDict):-
    name: Annotated[str, "Name of Author"]
    age: Annotated[int, "Age of Author"]
```

اس میں ہمارا model غلطی کر سکتا ہے کبھی کبھی کہ وہ str دے گا بجائے int کے جو اسکو دیا

Pydantic:- Simple:-
Pydantic is a data validation and data parsing library in python. It ensure that the data you work with is correct, structured and Type-safe.
Default, Optional, Coerce, Builtin, Field Function

```
from pdantic import BaseModel.
class Student(BaseModel)
    name: str = 'nitish'
    age: Optional[int] = None
    email: Email Str
    cgpa : Field(gt=0, lt=10, descrip"
new-student = {'age'= '32', email : 'ab'}
```

JSON-schema

```
{
    "title": "Student",
    "description": "                    ",
    "type": "object",
    "properties": {

    },
    "required": ["name"]
}
```

جب اور بھی language سے integrate کرنا ہو تو
JSON-sch کو استعمال کرتے ہیں۔

OpenSourceModel کے لیے ضروری استعمال نہیں کرتے ہیں
with-structured-output
یہ meta-llama ہے اس کے پاس
parameter نہیں ہوتے ہیں۔

Output parsers:-

Output-parsers help to convert raw LLM responces into structured format like JSON, CSV, pydantic models and more. They ensure consistency, validation and ease of use in application

1) StrOutputParser:-

اگر آپ یہ استعمال کرتے ہو تو آپ کو result.content نہیں لکھنا پڑتا۔ یہ ہمارا ماڈل خود سے remove meta-data کر دیتا ہے۔

کیوں یہ ضروری ہے جبکہ ہمارے پاس result.content ہے؟

Topic → LLM → [detaild report] → LLM → 5 line sumary.

```
parser = StrOutputParser()
chain = template1 | model | parser | template2 |
        model | parser
chain.invoke({"topic" : "Blace Hole"})
```

2) JSONOutputParser

```
Parser = JSONOutputParser()
template= PromptTemplate{
    template ='Give me the name, age and city of person, \n {
{format_instruction}',
```

input_variables = [ ],
partial_variables = {'format_instruction':
parser.get_format_instruction }

prompt = template.formate( )
result = model.invoke(Prompt)
final_result = parser.parse(result.content)
print(final_result).

JSON Output Parser does not give json-schema.

3. Structured Output Parser :-
is an output parser in LangChain that helps extract structured json data from LLM responses based on predefined field schema.
using ... schema ...

Response schema
["fail", description = "..."]

Disadvantage :-
- ... data validation ...

4) **Pydantic Output Parser**

is a structured outpparser that use pydantic models to enforce schema validation when processing LLM responses

data validation] - Pydantic
schemas]

Seamless integration = Works well with other Langchain components.

Easy Validation = Uses Pydantic's built-in validation to catch incorrect or missing data.

Video # 7

Chain → ہمیں اس میں صرف ایک بار input دینا ہوتا ہے۔

- Simple | Sequential Prompt | model | parser

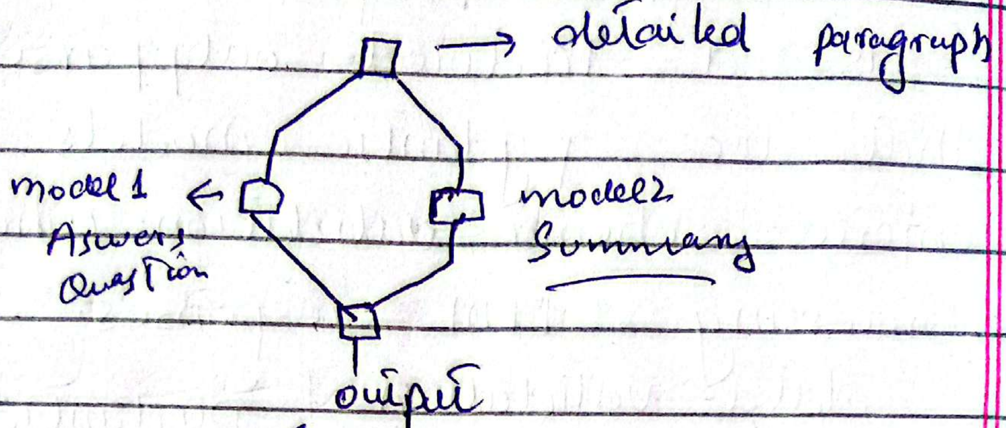
جس کی help سے pipeline بنا سکتے ہو۔ اور چھوٹے چھوٹے steps کو جوڑ سکتے ہو۔ جس میں ایک step کا output دوسرے step کا input بن جاتا ہے automatically۔

- Sequential :-

جہاں ہم دو بار model کو call کرتے ہیں۔ لیکن یہ بھی ویسا ہی simple ہوتا ہے۔ بس prompt میں لیں گے۔

LLM میں user ایک مکمل text دے گا "detailed"۔ اور ہم اس text کو 5-point کی summary میں تبدیل کریں گے۔

parrallel Chain



Runnable parallel جب ہے کہ ہم in parrallel chain
کی ضرورت پڑتی ہے

```
parallel_chain = RunnableParallel({
    'notes' : prompt1 | model1 | parser,
    'Ans' : prompt2 | model2 | parser,
})

merge_chain = promp3 | model1 | parser
chain = parallel-chain | merge-chain
chain.invoke({'text': text }).
```

اچھا یہ prompt1 جو ہے notes والا generate notes
کرے گا اور دوسرا prompt Answer generate کرے گا اور
Runnable parallel ان دونوں کو merge کرے گا اور
output جو تیسرے prompt میں بھیج دے گا جس
میں already ہم نے notes اور queries
input variable دے رکھا ہے اس کی input
جب کہ گا اور پھر last chain تیسرے
prompt کے output نکال کر دے گا۔

Conditional Chains

کسی conditions کی بنا پر بنتے ہیں۔ کہ اگر یہ کام ہوا تو یہ chain چلے گا اور اگر ایسا ہوا تو یہ chain چلے گا۔
اس کے لیے ہم RunableChain کا استعمال کرتے ہیں۔
کسی Feedback کی بنا پر message کرنا کہ اگر feedback negative ہے تو یہ message جائے گا اور اگر positive ہوا تو یہ message جائے گا۔

= RunnableChain(
 (, chain),
 (, chain),
 RunnableLambda( )

Runnable کے بغیر chains کام نہیں کر سکتے۔
chains کو چلانے کے لیے آپ کو runnable کا concept پتا ہونا چاہیے۔

Runnable کی ضرورت کیوں پیش آئی۔

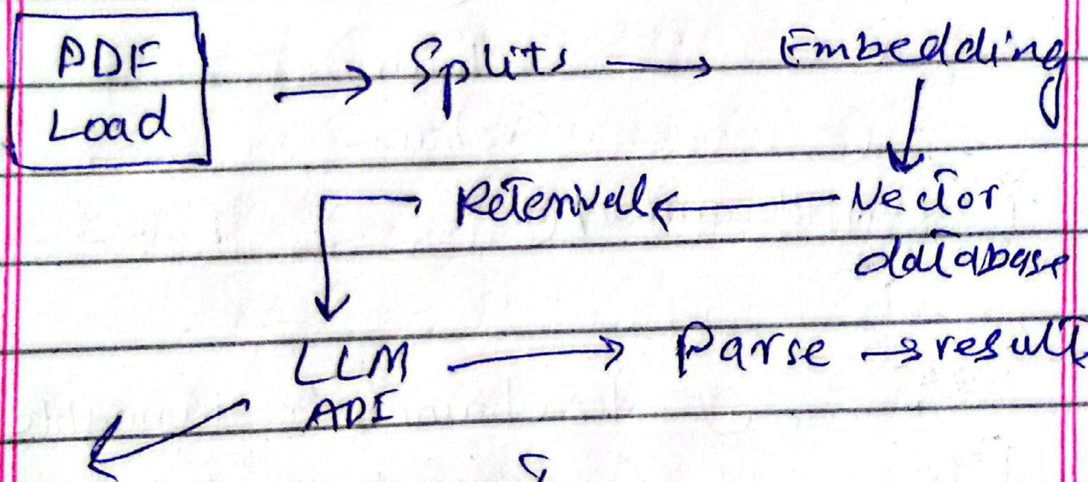
2022 → ChatGPT realize
→ OpenAI API → build LLM application

AI Agent, ChatBot, PDFReaders بننے لگے تو تب پہلے سے زیادہ
کر دی تو LangChain کی ٹیم نے یہ سوچا کہ کیسے ہم developers کی help کر سکتے ہیں۔ تو انہوں نے یہ بات identify کی۔ ہر company اپنے LLM's بنا رہی ہے تو ایسا ...

LLMs کی API company کی بنائی ہیں ہم کسی framework
سے صرف اس سے بات کر سکتے ہیں۔ یہ سب سے بڑی problem
solve کی۔

دوسری problems کو langchain نے
identify کیا جب ہم application بنا رہے ہیں تو
صرف API سے interact کرنے کے علاوہ
اور بھی بہت سارے task ہیں۔ For example:

PDF Reader



its only one component.

تو انہوں نے باقی کے task کے لیے الگ الگ
component بنانا شروع کر دیے۔ جیسے کہ pdf
parse PDF Loader component بنا دیا۔
کے لیے parser بنا دیے۔ تو developer آسانی
سے application complex کو بھی بنا سکتے ہیں۔
AI Engineers نے ان component کو استعمال
کیا اور application بنانے لگے۔
تو اس کے دوران langchain نے ایک اور
بات notice کی۔ کہ ہر application میں کچھ

کام دوبارہ دوبارہ ہو رہے ہیں۔ جیسے prompt بنانا
پھر اس میں call کرنا اور اور llm کو call کرنا
تو انہوں نے ہم ایسا بنا دے builtin fun جس
میں آپ صرف llm, prompt کو call کر دو اور
وہ result لا کر دے دے۔

LLMChain ~~chain~~ (llm, prompt)

دو یا دو سے زیادہ component کو جوڑ رہے ہو۔

اچھا ہے تو ایک simple task تھا langchain سے
complex chain بھی بنا سکتے ہیں۔ جیسے RAG applic...
میں retrieval کا task ہوتا ہے۔ آپ user سے
query لیتے ہو اور پھر documents میں same page اٹھاتے
ہو اور اور ان دونوں کو merge کر کے llm
کو call کرتے ہیں۔ team نے notice کیا اور
ایک function بنایا جس میں آپ query دے اور
retrieval کو اٹھاتے ہو اور chain میں ڈال دیتے ہو۔
اور result automatically آ جاتا ہے۔

RetrievalQAChain (retrieval, ~~query~~ llm = llm)

تو انہوں نے پھر خوب سارے chain بنانا شروع
کر دیے۔ جہاں بھی ان کو لگا کہ task
reuse ہو رہا ہے تو اس کے لیے انہوں نے chain
بنا دیا۔

SimpleSequentialChain

llm $\xrightarrow{\text{call}}$ joke $\xrightarrow{\text{llm call}}$ Explanation.

تو langchain کی team نے 50+ chain
بنا دی۔ تو اس وقت تک ان کو یہ idea نہیں
تھا کہ وہ بہت بڑی problems میں پھنسنے والے ہیں۔
ان کا intent یہ تھا کہ وہ AI Engineer کی
help کر سکیں کیونکہ ان کے پاس بہت سارے
component تھے اور پھر ان کو plug کر سکیں۔
تو problems یہ آئی کہ بہت سارے chains کی وجہ سے

→ database huge

→ new learner

کے لیے بہت سارے chain ہونے کی وجہ سے
یہ سمجھنا مشکل ہو گیا کہ کون سے task کے لیے
کون سا chain استعمال کرنا ہے۔

اس کی سب سے بڑی وجہ یہ تھی کہ انہوں نے
بہت سے component بنا دیے تھے۔ i.e. parser
تو standardize نہیں تھے یعنی کہ ان کے common method
نہیں parser کو call کرنے کے لیے parse
کا استعمال کیا جاتا تھا۔ اور prompt template کو call کرنے
کے لیے format کا استعمال کرتے تھے۔ llm
کو call کرنے کے لیے آپ predict استعمال
کرتے تھے۔

تو انہوں نے اس طریقے سے design
کیا کہ components کا ہر method abstract
تو call کرنے کے لیے آپ کو invoke کا استعمال
کرنا پڑے گا۔ اور جب standard component بن گئے تو
تو پھر chain بنانا آسان ہو گیا۔

(Runnables)

[Task Specific Runnables]

- These are the components like Prompt Template, llm, parsers

standards
↓
Runnables.

[Runnable Primitives]

For connecting task specific components

Prompt ⟷ LLM
input —output→ input

- Runnable Sequence
- RunnableParallel
- RunnableLambda

RunnableSequential

اس کی help سے دو یا دو سے زیادہ runnable کو
connect کر سکتے ہیں۔ تو یہ chain one-by-one
ں۔ جائے گا اور اس کا output دوسرے کا input
میں جاتا ہے

- RunnableSequential(prompt, llm, parser)

Runnable Parallel:-

topic AI
↙ llm1 → ↓ tweet
↘ llm2 → ↓ linkedin

یہ دونوں chain parallely execute ہوتے ہیں اور ان کا input same ہوتا ہے

RunnableParallel ({
'tweet' : RunnableSequence(prompt, model, parser),
'linkedInPost' : RunnableSequence(prompt, model)
}),

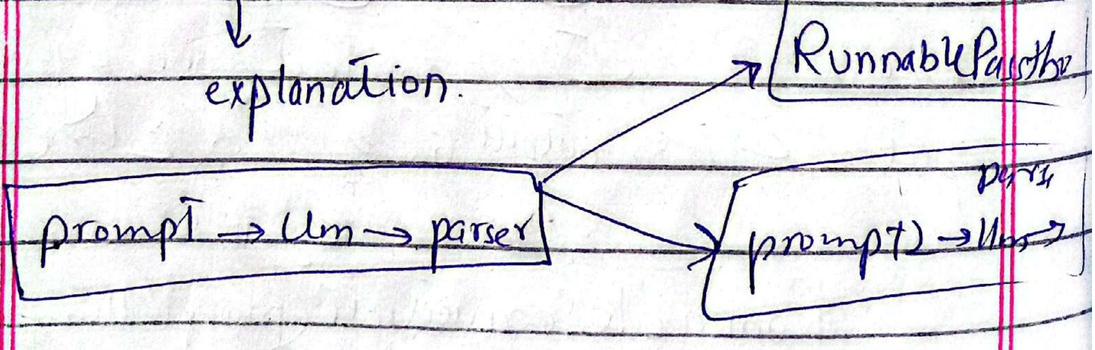
RunnablePassthrough

is a special Runnable Primitive that simply returns the input as output without modifying it.

جو آپ کا input دیں گے وہی آپ کو
output میں دے دے گا۔

For example — my requirement

topic : AI
↓
joke ✗ → joke + explanation
↓
explanation.



RunnableLambda

اس کا استعمال ہم اپنے llm سے extra
اس میں ہم کچھ کرنے کے لیے generate output
اس میں ہم Lambda func کا استعمال کرتے
کوئی func manual لکھ سکتے ہیں

RPC {

'word count': RunnableLambda(lambda x: len(x.split()))

RunnableBranch → (if no condition match, default runnable is used)

(you define set of conditions) ← for conditional Chain

if-else of universe of Langchain

"is a control flow component in LangChain that allow you to conditionally route data to different chains or runnable based on custom data. e.g

→ For complaign —send→ custom team
→ refund —send→ database
→ general query —send→ chatbot.

email → prompt → llm → RB → complain / refund / general.

anyone branch will trigered.

To execute branch کسی چیز کے ہونے پر کوئی ایک

LCEL (|)

Langchain کی team نے سوچا کہ کچھ چیزیں جو runnable میں دوبارہ دوبارہ استعمال ہو رہی ہیں جیسے runnable sequence تقریباً ہر جگہ runnable میں استعمال ہوتا ہی ہے تو کیوں نا ہم اس کو easy بنائیں تو انہوں نے ایک لائن لکھی کہ کیا آسان کر دیا

~~RunnableSequence (prompt, model, parser)~~ ✗

✓ = prompt | model | parser

Vedio # 11:-

Document Loader

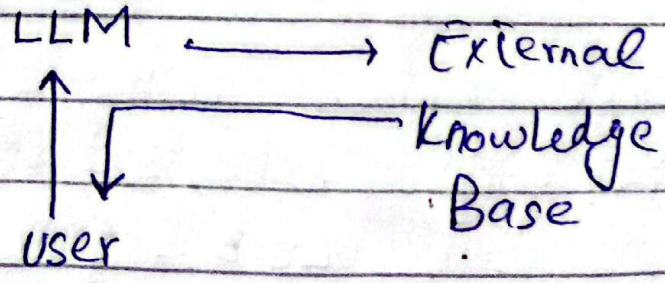
- RAG:-

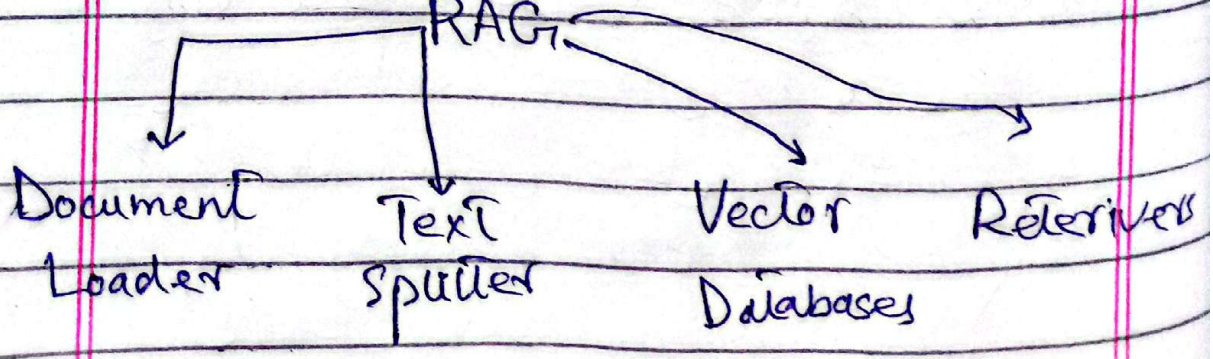
is a technique that combines information retrieval with language generation where a model retrieves relevant documents from a knowledge base and then use them as context to generate accurate and grounded responses.

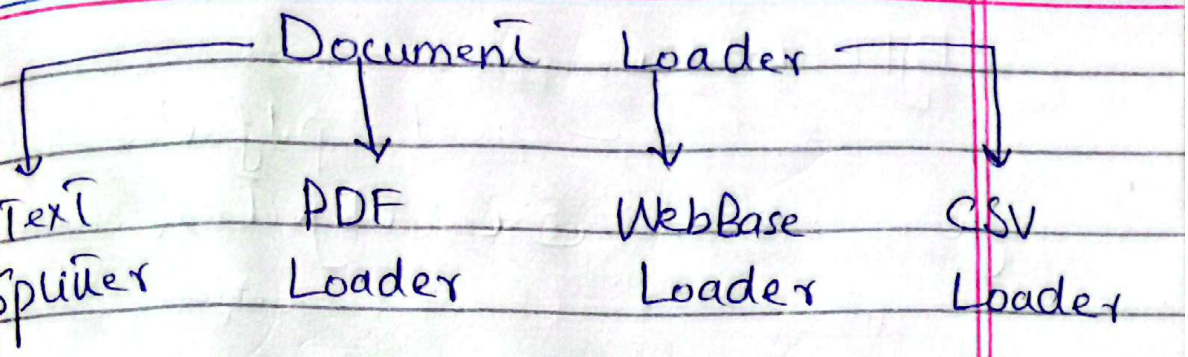
1). Use of up-to-date information
2). Better privacy
3). No limite of document size

Eexample

Chatbot → Chatgpt
X current affaires
X personal data







Document Loader are components in LangChain used to load data from various sources into a standardized format (usually as Doument object), which can then be used for chunking, embedding, retrieval, and generation.

```
Document(
  page_content="The actual text content",
  metadata={'source': "file_name", ....}
)
```

① TextLoader

is a simple and commonly used document loader in LangChain that reads plain text files (*.txt) and converts them into Langchain document.

chat logs
transcript
output in [ ]
list
.scraped text
. code snippets

PyPDFloader

یہ آپ کی pdf کو read کرتا ہے اور ہر page کو ایک document object میں بھیج دیتا ہے۔ یہ ہر page کو بھیجتا ہے۔

یہ list ہوتی ہے۔

```
[
    Document(page_content="Tex from page1", metadata={}),
    Document(page_content='page 2', metadata={page=1}),
    ....
]
```

Limitation :-

یہ جب آپ اسے pypdf میں بھیجو گے اپنے page کے pdf کے کو تو وہ document کے بنا دیتا ہے۔

simple text pdf کے لیے ہے۔

تصویر والے pdf کے لیے الگ pdf loader ہیں۔

UseCases	Recommended Loader
Simple, clean PDF's	PyPDF Loader
PDF's with tables/column	PDFPlumber Loader
Scanned/Image Pdf's	UnstructuredPDFLoader/AmazonTextract
Need LayOut Image Data	PYMUPDF Loader
Want best structure extraction	UnstructurePDFLoader

DirectoryLoader

یہ مکمل directory کو load کرنے کی صلاحیت
رکھتا ہے۔ یعنی اگر آپ نے کسی directory میں
میں سارے pdf ہے یا سارے txt ہے
تو load کرنا ہے تو آپ directory کو استعمال
کر سکتے ہیں۔ آپ نے بتانا ہوتا ہے اگر آپ
کے پاس pdf ہیں load کرنی ہیں تو آپ نے
pdf PyPdf دینا ہے loader_cls میں۔

```
DirectoryLoader(
    path='',
    glob='*.pdf',
    loader_cls='PyPDF',
)
```

→ you also give '*.txt' → TextLoader

Load( )	Lazy_Load( )
• Eager Loading (loads everything)	• Lazy Loading (loads on demand).
• Return a list of document object	• Return a generator document
• loads all documents immediatly into memory	• Document are not all located at once: they're fetched once at a time as needed.
• Best when:-	Best when:-
• The no. doument is small	You are loading with large document
• You want everything load upfront	

WebBaseLoader :-

is a document loader in Langchain used to load and extract text content from web pages (URL).

It uses Beautiful soupe under the hood to parse HTML and extract visible text.

- When To Use :-

For blogs, news article, or public Websites when the content is primarily text-based and static.

- Limitations :-

Don't handle JavaScript-heavy pages well (Use SeleniumURLLoader for this).

WebBaseLoader:

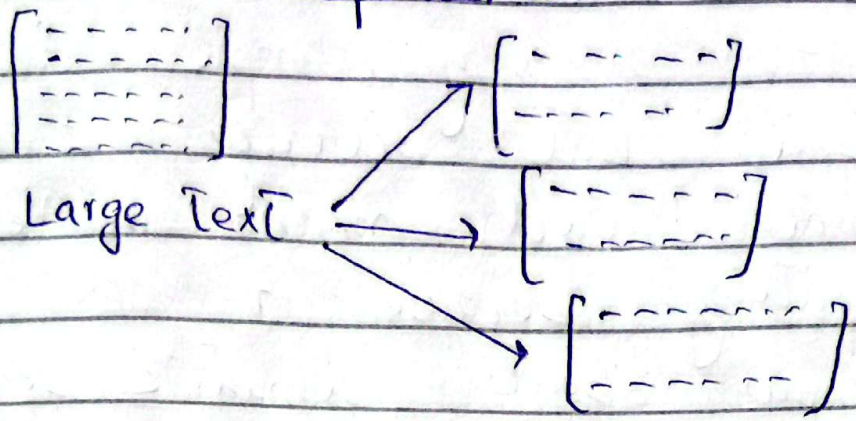
url کے page کے website کی اس میں آپ کی دیتے ہیں۔

CSV Loader.

اس میں آپ path کے csv دیتے ہیں، اور یہ document کی rows

Video # 11:

Text Splitter:-

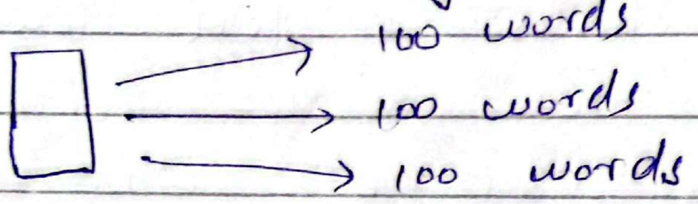


is a process of breaking large chunks of text (like Articles, pdf's) into smaller and manageable pieces that LLM can handle effectively.

- Overcoming model limitation:-
  Many language model and language model has maximum input size constraints. Splitting allows us to process document that would otherwise exceed this limit.
- Downstreaming Task:- Text splitting improves every LLM powered task.
  i) Embedding
  ii) Semmantic Search
  iii) Summarization.
- Optimizing computational Task.

— Length Base Text Splitting

It is very simple and fast but actually it is not used much in text splitting because it does not know about semantic search even not know grammer.



Length Base Splitter Text

↙ Character — after 100 words :

↘ Tokens

Text Structured Base

RecursiveCharacterTextSplitter

\n\n , \n , ' ' , ''

para , Line , space , character

Allowed

chunk-size=10

اس para کی base پر اگر chunk-size 10 ہے

تو تمام para میں جو کہ chunk-size 10 سے بڑا ہے

تو پھر line کی base پر کریں گے

اور پھر یہ اسی طرح سے space کی base پر اور پھر character کی base پر ۔

Benefits:

یہ ہمیشہ try out کرتا ہے کہ مکمل para یا مکمل sentence split کرے ۔

- Document-Structured Based
  - For piece of code
  - For markdown.

"\nclass", "\ndef" — for class, for function — First step

\n\n, \n, ' ' — para, line, space — Second step

Semantic Meaning Based

اس میں یہ دیکھا جا رہا ہے کہ semantic

sentence — text-splitting based کرتا ہے ۔

S1	S2	S3	S4	S5
0.9	0.8	0.85	0.1	

sematanity — embedding model.

یہ ایک sliding window استعمال کرتے ہیں

اور جہاں semalarity کم ہو وہاں Text Splitting کرتے ہیں ۔

پھر ہم ان کا semalarity-score کا std (?) نکالتے ہیں، اور اگر یہ 1 ہو تو ہم Text-splitting کرتے ہیں ۔ یہ embedding models لگاتے ہیں اور پھر semantic search

Vector Stores

Embedding سے جو vectors بنتے ہیں انہیں store کرنے کے لیے استعمال کرتے ہیں۔ ہم vectors کو functional databases میں نہیں رکھ سکتے۔ کیونکہ پھر ہم ان کے ساتھ semantic search نہیں کر سکتے۔

A vectore store is a system designed to store and reterieve data represented as numerical vectors.

- Storage :- Ensure that vectors and their associated metadata are retrained. whether in-memory and on-disk.
- Similarity Search :-
  Help reterieve the vectors most similar to query vector.
- Indexing :- Provide a data structure or method that enables fast similarity searching on high-dimensional vectors (ie approxiate nearest neigbour lookups)
- CRUD Operation :-

Use Cases:-
i) Semantic Search 2. RAG
iii) Recommendation System
iv) Image / Multimedia Search.

- A vector database is effectively a vector store with extra database features (e.g clustering, scaling, security, metadata filtering and durability)

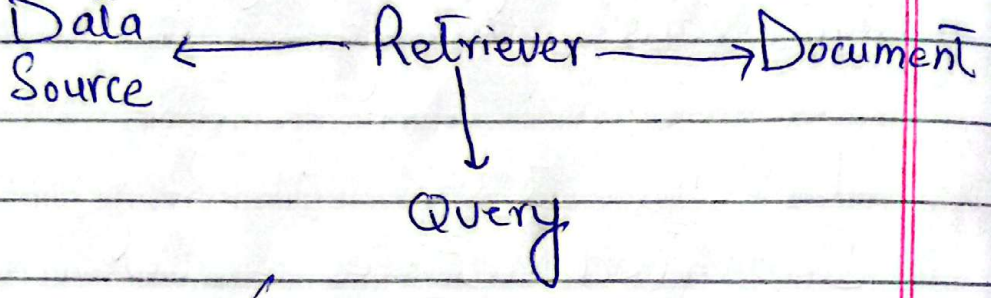
- Vector Stores in LangChain:-
- Supported Stores :- Langchain integrate with multiple vector stores (FAISS, Chroma ie).
- Common Interface
- Metadata Handling:-
  Most vector stores allows you to attach metadata. ie Timestamp, author.
- Chroma:-
  Chroma is a lightweight, open-source vector database that is especially friendly for local development and small to medium scale production use.

Video no # 13

(Retrievers)

A retriever is a component in Langchain that fetches relevant ~~component~~ documents from a data source in response to a user's query.

All retrievers are runnables.



آپ retriever سے بات کرتے ہیں۔
وہ data source سے documents لاتا ہے۔
پھر وہ searching کا طریقہ اپناتا ہے۔

— Wikipedia Retriever

is a retriever that queries the wikipedia api to fetch relevant content for a given query.

= How it works :-

- You give it a query (e.g. "Albert Ein...
- It sends the query to wikipedia api.

- It retrieves the most relevant articles.
- It returns them as Langchain Document object.

retriever = WikipediaRetriever(top_k_result=2, lang='en')

- VectorStore Retriever:-
  is the most common type of retriever that let you search and fetch document from vector store based on simantic similarity using vector embedding

= How it works.

- You store your document in vector store (FAISS, CHROMA, Weaviate)
- Each document is converted into dense vector using embedding model.
- When the user enter query:-
  - It also Turn into a vector.
  - retriever compare the query vector with the stored vector.
  - It recieves the Top-K-most similar ones.

= Mimimal Marginal Relevance

"How can we pick results that are not only relevant to the query but also different from each other."

MMR is an information retrieval algorithem designed to reduce redundancy in the retrieved results while maintaining high relevance to the query.

= Why MMR Retriever:-

In regular similarity search, you may get document that are:

- All very similar to eachother
- Reapting the same info.
- Lacking diverse perspective.

MMR retriever avoids that by:
- Picking the same relevant document first
- Then picking the next most similar and least similar to already selected docs:

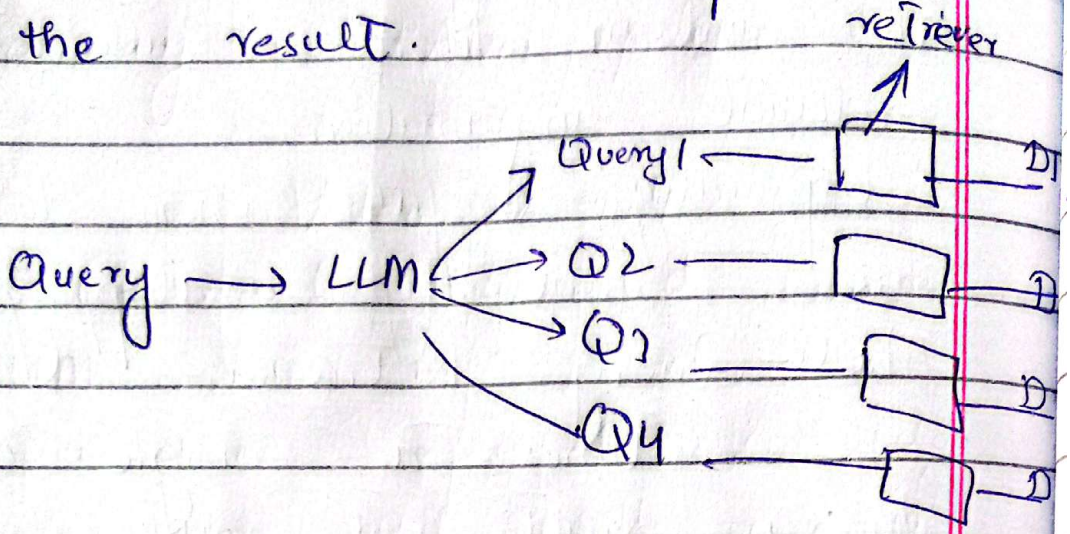
and so on....

This helps especially in RAG pipelines where:
- You want your context window to contain diverse but still relevant information.
- Especially useful when documents are semantically overlapping.

– Multi-Query Retriever:-
"Sometimes a single query might not capture all the ways information is phrased in your document."
Query:-
"How can I stay healthy."
Could means:-
- What should I eat.
- How often should I excercised.
- How can I manage stress.

A simple similarity search might miss documents that talk about those things
- Take your original query.
- Use an LLM to generate multiple siminatically different versions of that query.

- perform retrieval for each subquery.
- combines and deduplicates the result.



— Contextual Compression Retriever is an advanced retriever that improves retrieval quality by compressing document after retrieval - keeping only the relevant content based on the query.

? 'Query'

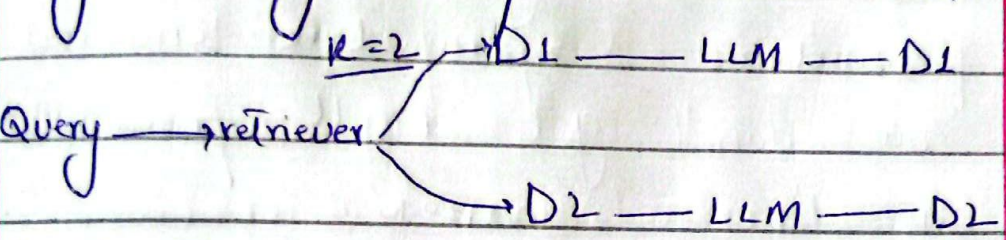
'What is photosynthesis?'

Retrieval Document (by traditional retrieval)

"The grand Canyon is famous natural site. photosynthesis is how plants convert light into energy. Many tourists visits every year."

Problem :-

. The retriever returns the entire paragraph.
. Only one sentense actually relevant to the query
. The rest is irrelevant noise that waste context window any may confose the LLM.



Contextual Compression Retriever does:-
. Returns only the relevant part "photosynthesis is how plants convert..."

= How it works :-

1. Base Retriever (e.g FAISS) retrieves N-Documents
2. A compressor (LLM) is applied to each document.
3. The compressor keeps the part only relevant to query.
4. Irrelevant content is discarded.

= When To USE :-

Your document are long and contain mixed information.

- You want to reduce context length for LLM's.
- You need to improve answer accuracy in RAG pipelines.

Video no #14

(Retrieval Augumented Generation)

- LLM store information in parameters → parametric knowledge
- You can't get result from LLM on these scenario's.
  - i) Private data.
  - ii) Recent Information
  - iii) Hulucination.

– Solution → fine Tunning.
pretrained the LLM → again Train on smaller data → domain specific

fine Tuning.

- Supervised ufise fine Tunning
- (ontinue pretraining
- RLHF

= Problems Again on Fine Tuning
- i) Training on large data computation expertise
- ii) Technical experts.

Solution:- Again

In context Learning:-

solve Taske by seeing examples in prompts without updating its weights.

Prompt Example:-

Please sentiment these:
I love my school → possitive.
This app crashes a lot → negetive.
I hate movies →

↓

few shot prompting

It is an emergent property.

اگر کسی prompt میں examples دے دیں تو بغیر training کے
context سے سیکھ کر llm model اپنا
improve کر لیتا ہے۔

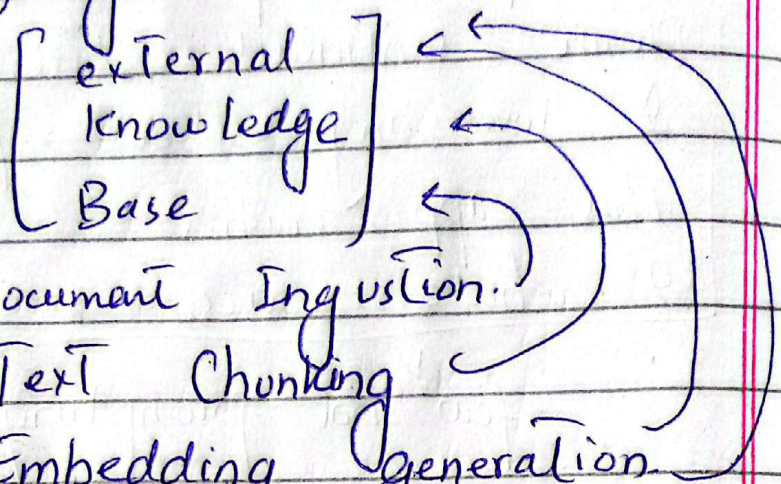
RAG is to make a language model smarter by giving it extra information at the time you ask the question.

— Understanding the RAG:-

i) Indexing
ii) Retrieval
iii) Augmentation
(iv) Generation

— Indexing :-

is a proess of preparing your knowledge base so that it efficiently searched at query time.

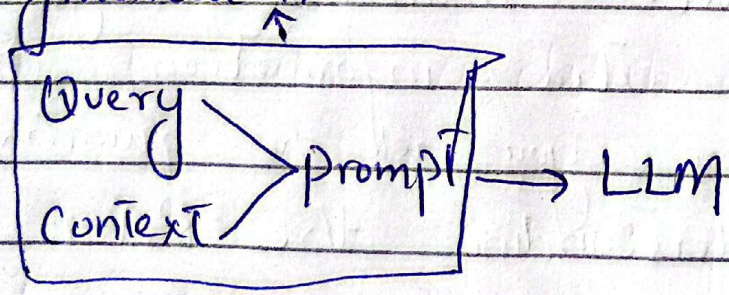


i) Document Ingestion.
ii) Text Chunking
iii) Embedding generation.
iv) Storage in vector store

— Retrieval

finding the most relevant pieces of information from vector store.

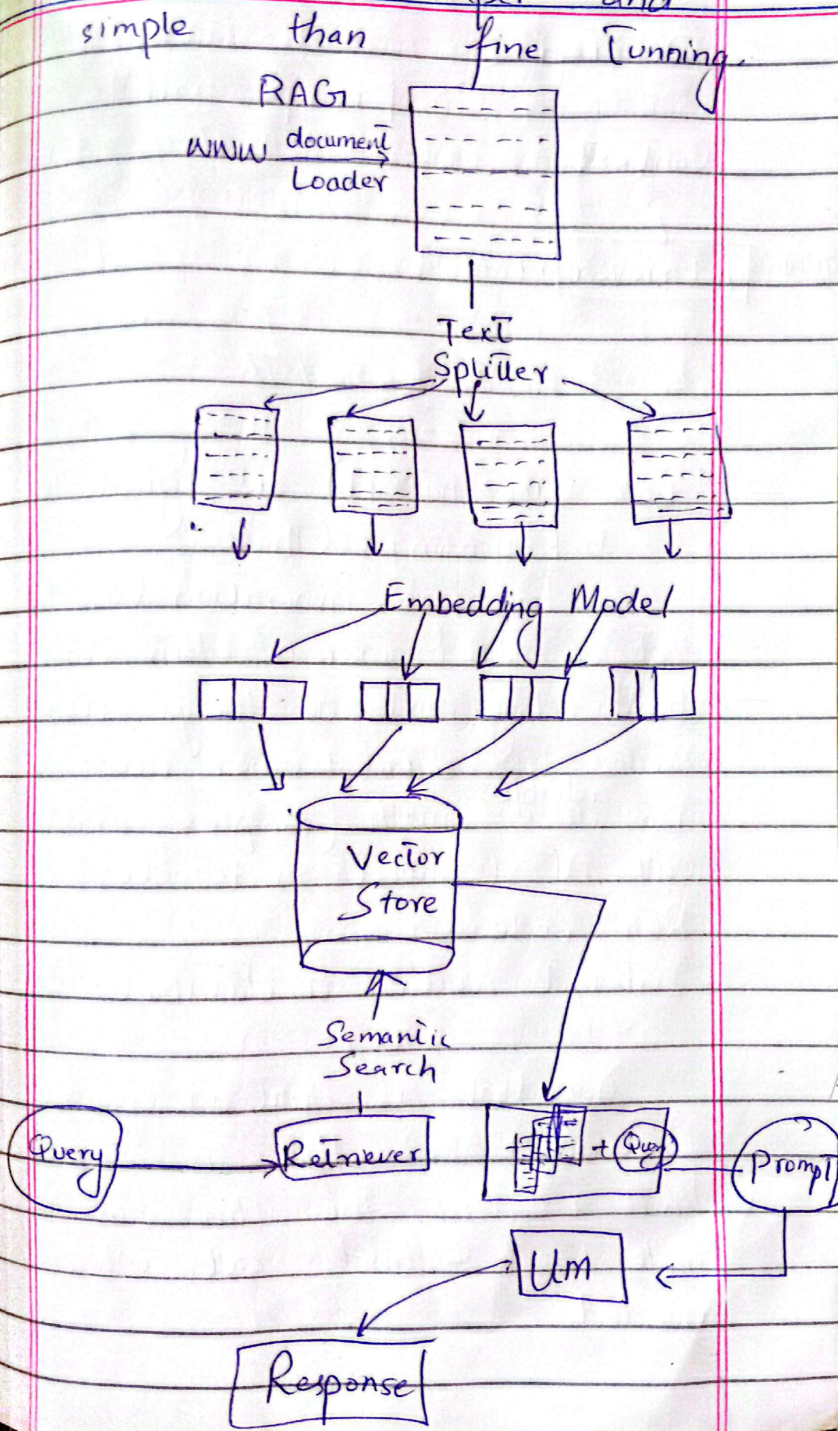
— Augmentation :-

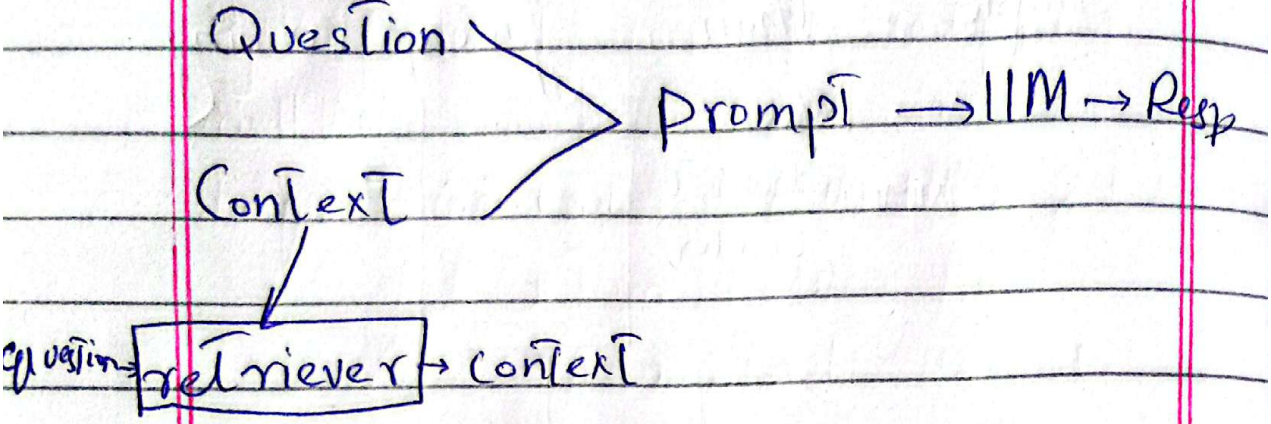


— Generation

prompt → LLM → Response

Now RAG solve all problem and it is Cheaper and simple than fine Tunning.





Video no# 16

Tools

— LLM (like ChatGPT) are great at:-
  i) Reasoning (Think)
  ii) Language generation (speak)
that is all power of llm only.

— But they can't do things like:-
- Access live data (weather, news).
- Do ~~accu~~ reliable math (complex math)
- Call APIs (tweet on tweeter)
- Run code
- Interact with a database.

A tool is just a python function (Or API) that is packaged in a way the llm can understand and call when needed.

Tools

Built-in-Tools — Custom Tools

How Tools fits into the Agent ecosystem.

"An AI Agent is an LLM-powered system that can autonomously think, decide and take actions using external Tools or API's To achieve a goal."

Agent

Reasoning / Decision making → LLM

Actions → Tools

Built-in-Tools:-

A built-in Tool is a Tool that LangChain already provide for you — it's pre-built, production ready, and requires minimal or not setup.

You don't have To write the function logic yourself, — you just import and use it.

- DuckDuckGo Search Run
- Wikipedia Query Run

These Tools are also runnable.

- PythonReplTool — Run raw python code
- ShellTool — Run shell commands.
- RequestsGetTool — Make HTTP get requests

Custom Tools

A custom Tool is a Tool that you define yourself. Use them when:

- You want to call your own API's
- You want to encapsulate business logic.
- You want the LLM to interact with your databases, product or app.

from langchain core tools import Tool.

- create function
- doc string
- type hint
- add decorator.

@Tool make this a special function that can communicate with LLM.

جب اس tool کو LLM کے پاس بھیجتے ہیں تو
اسے tool نہیں دکھائی دیتا بلکہ اس کا [illegible] json
schema دکھائی دیتا ہے

Ways To create custom Tool:-
(i) using aTool (ii) using StructuredTool decorator & pydantic
(iii) Using BaseTool Class.

- "A structured Tool in Langchain is a special type of Tool where the input to the Tool follows a structured schema, typically defined using a Pydantic model."
- "BaseTool is the abstract base class for all Tools in Langchain. It defines the core structure and interface that any Tool must follow, whether it's a simple one-liner or a fully customized function."

All other Tool types like aTool, StructuredTool are built on Top of base Tool.

- Toolkits:-

A toolkits is just a collection of related Tools that serve a common purpose -

best benefit.

package together for convenience and reusability

i) GoogleDriveCreateFileTool
ii) GoogleDriveSearchTool

GoogleDriveToolkit.

Video no#17

Tool Calling

- Tool Binding
  - Tool B کے ذریعے llm کو یہ بتانا کہ
    اس کے پاس کون کون سے tool موجود ہیں۔
    دوسرا وہ tool کون کون سے task کر سکتے ہیں۔
    تیسرا وہ tool کیا input expect کرتے ہیں۔

- Tool Calling:-
  is the process where llm decides during a conversation or task that it needs to use a specific tool — and generate a structured output with
  - the name of the tool.
  - and the arguments to call it with.

* The llm does not actually run the tool. It just suggest the Tool and the input arguments. The actual execution is handle by you or Langchain.

- Tool Execution:-

Tool Execution is the step where the actual python function is run using the input arguments that the LLM suggests during Tool calling.

Agenda - 15-Aug-2025
i) Problem solving of Tool with chat models.
ii) FAQ's Chat Assistent
iii) House price prediction
iv) Pakistan House price prediction updating.

Video no # 20

AI - Agents in Langchain

problem:-

آپ نے trip پر جانا ہو تو آپ web پر جا کے
پھر بہت سارا کام کریں گے جو کہ ایک مشکل کام ہے۔
AI agent ایک ایسا integent system ہے جسے آپ ایک
high level goal دیتے ہو۔ اُس کے پاس خود کا
دماغ ہوا اور external Tools کو call کر سکتا ہے۔
API or knowledge source کو hit کر سکتا ہے۔
اور وہ context کو یاد maintain کرتا ہے۔

Agent = llM + Tools

resoning mechanism

API, database, Search Tools

AI - Agents:-

i) ~~AI~~ Goal - driven (v) Adaptive rethinking
ii) Autonomous planing
iii) Tool using
(iv) Context Aware
(Maintains memory across steps).

Steps:- LLM + Tool

create-react-agent, AgentExecutor
hub.
prompt = hub.pull("hwchase17/react")
Agent = create-react-agent (llm, tools, prompt)
↳ is the main guy → for reasoning
= AgentExecutor (agent, tools, verbose)
↳ work actually. or execute the agent.