Exploring LangChain:
Revolutionizing the Future of AI and Language Models

What is LangChain?

An open-source framework that simplifies the development and deployment of applications powered by LLMs.

- Chain multiple LLMss together for multi-step reasoning
- Integrate external data sources like APIs and databases (RAG– Retrieval Augmehed Generation)
- Build dynamic workflows that are contextually aware
- Automate tasks and decidson-making with agents

A Real-World Example:

Building a customer support chatbot using LangChain:

- Chain models together for multi-layered processing (e.g. NLP and sentiment analysis)
- Use memory to remember customer interactions, creating a personalized experience
- Let indexes to fetch relevant data from your knowledge b

6 Key Components of LangChain

Models

The backbone of LangChain

Prompts

are instructions or querules provided to models

Chains

a sequence of operations where each step

Indexes

help optimize retrieval of relevant information

Memory

Allows LangChain applications to "remember" prior intera-ctions or states

Agents

autonommous decision-making systems
(e.g., creating icket, pulling knowtedge articles, etc.)

LangChain Alternatives?
LlamaIndex HayStack

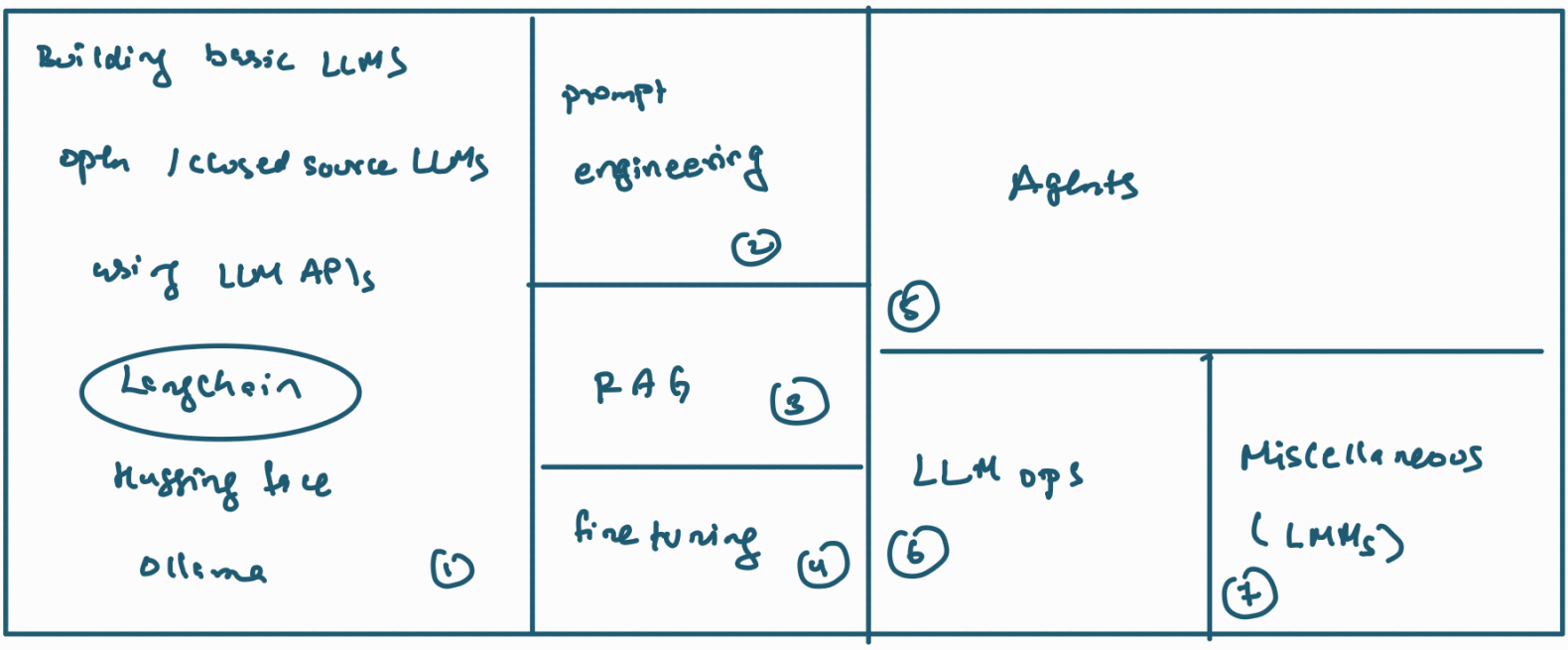
What is Langchain?

Langchain is an open source framework that helps in building LLM based applications.

It provides modular components and end-to-end tools that help developers build complex AI applications such as chatbots, question-answering systems, retrieval Augmented generation (RAG), autonomous agents and more.

i) Supports all major LLMs
ii) Simplifies developing LLM based Applications
iii) Integrations available for all major tools
iv) Supports all major GenAI use cases.
v) Open source (free) Actively developed.

GenAI
with Langchain Roadmap



Langchain

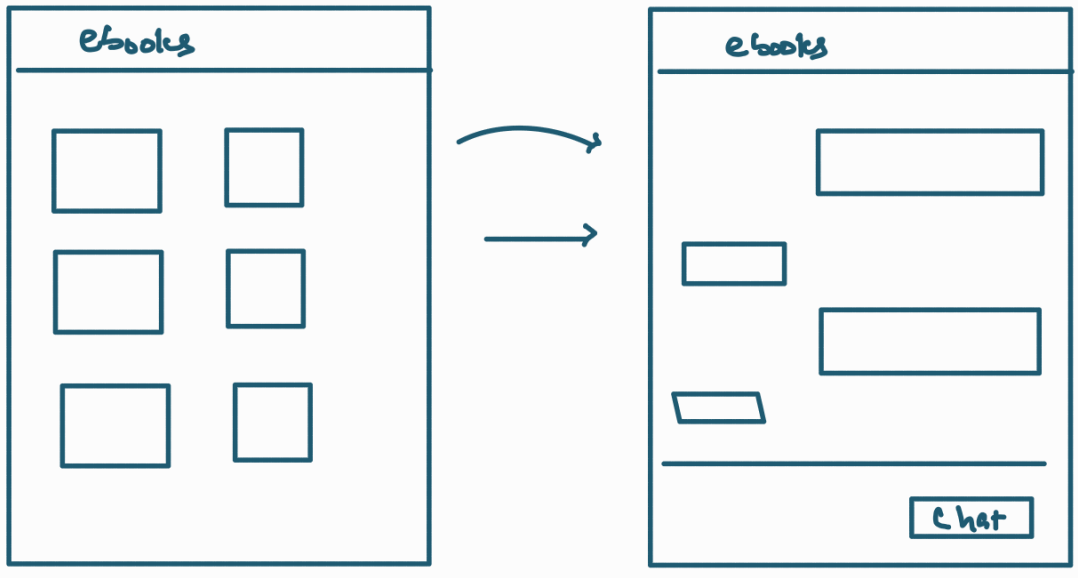
- **Fundamentals**
  1) what is langchain
  2) Langchain components
  3) Models
  4) Prompts
  5) Parsing output
  6) Runnables & LCEL
  7) Chains
  8) Memory
- **RAG**
  - Document loaders
  - Text splitters
  - Embeddings
  - vector stores
  - retrievers
  - building RAG applications
- **Agents**
  - Tools & toolkits
  - tool calling
  - building Ai Agent

Intro to Langchain :

→ Langchain is an open source framework for developing applications powered by LLMs

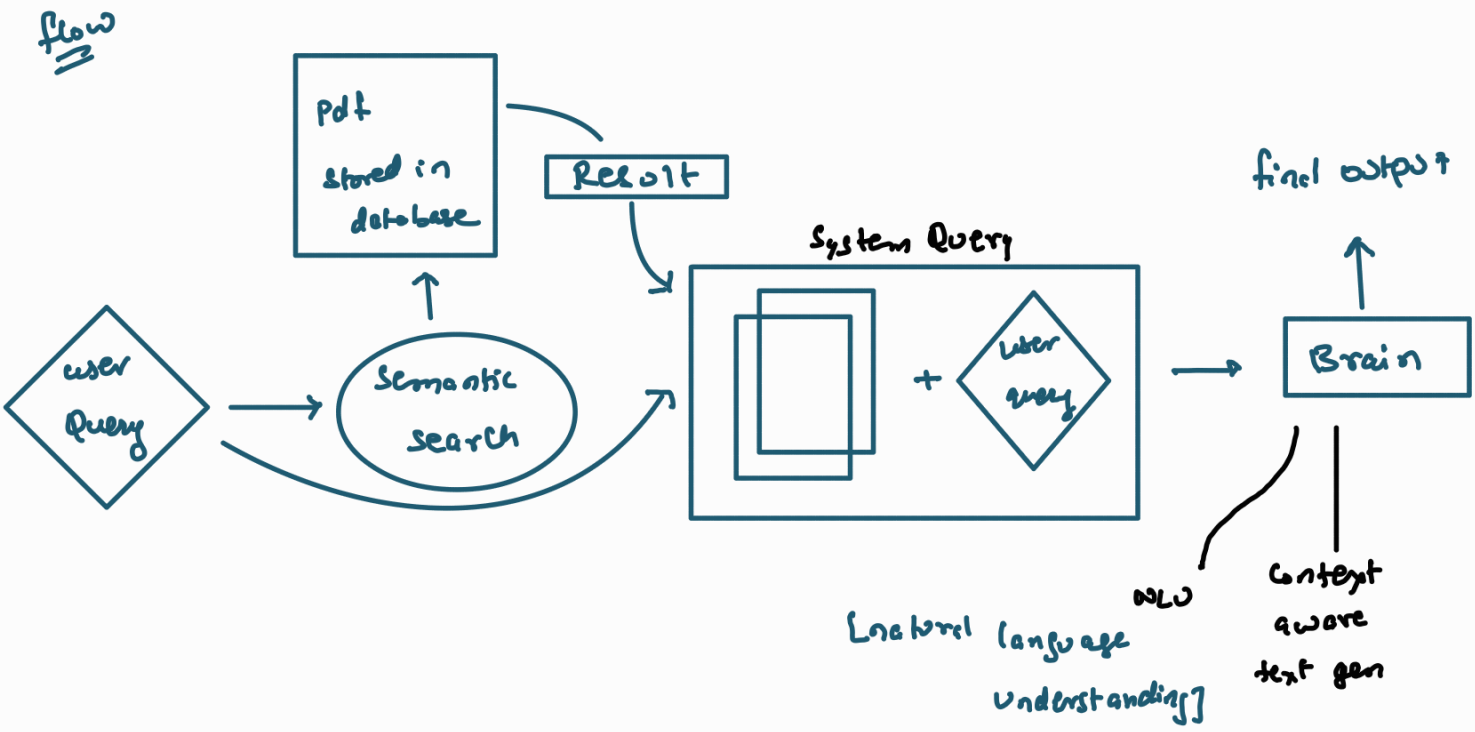
Why Langchain? To make Ai applications faster, scalable, model agnostic

eg.



to create an application to chat with ebooks directly.

chat with ebooks



Semantic search → search based on meaning of text

how many runs did virat score

↑

embedding

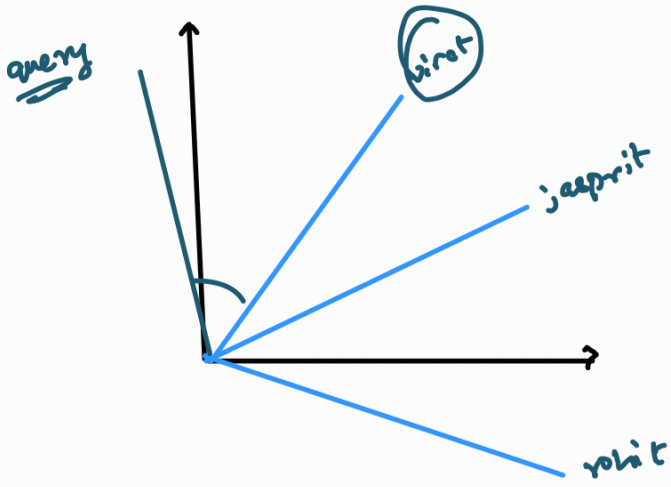
Paragraph about virat

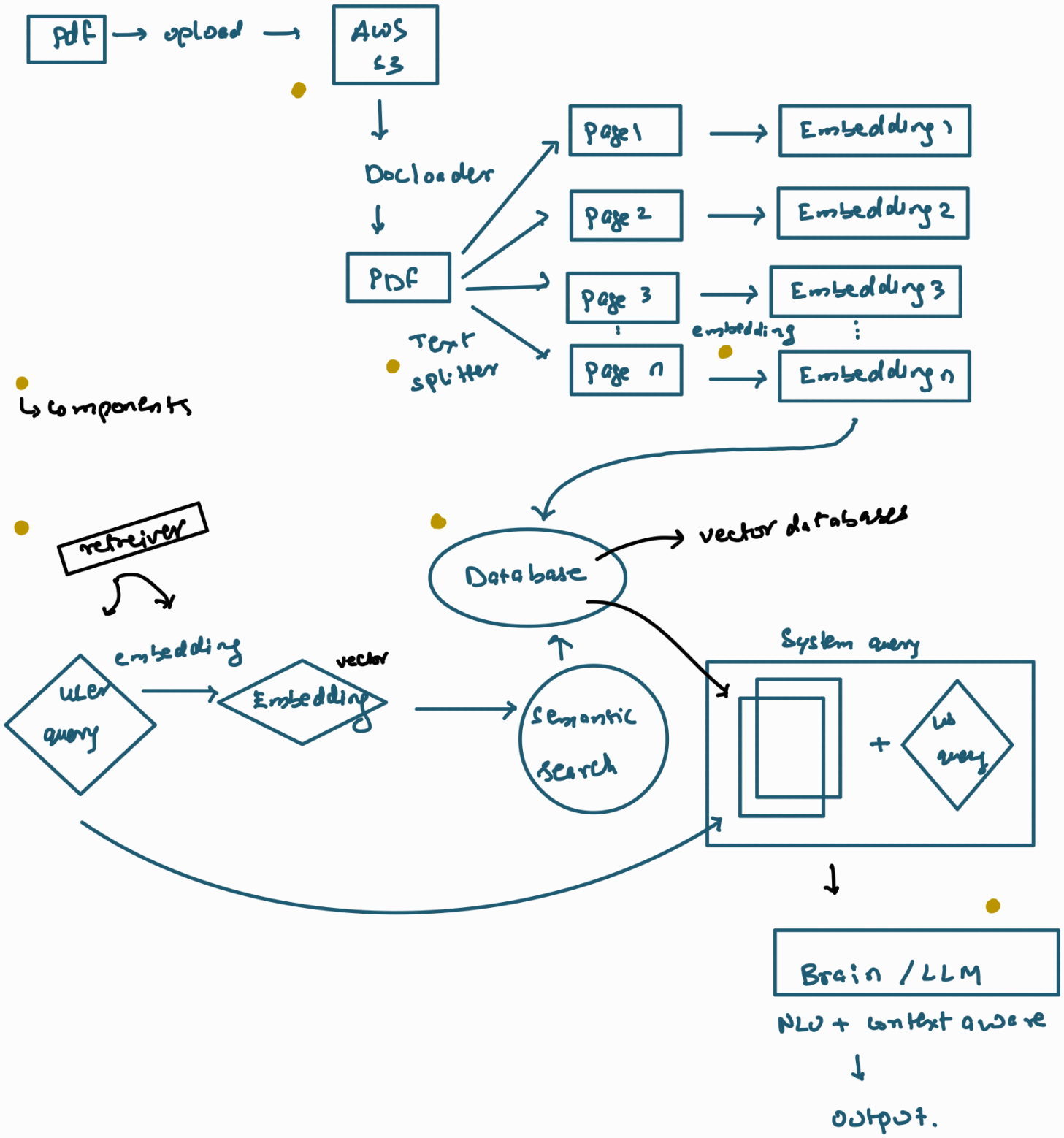
Paragraph about Jacprit

Paragraph about rohit

convert to embeddings (ie vectors)

through → word2vec
doc2vec
BERT





Challenges:

i) building the brain. (nlu & text gen)

↳ use LLMs

ii) LLMs models are heavy (computation heavy)

↳ we can use the wrapper API

iii) Orchestration

↳ use Langchain

Benefits

i) Concept of Chains

ii) model agnostic development

iii) complex ecosystem (orchestration)

iv) memory & state handling

What can be build?

Conversational chatbot

Ai knowledge Assistants

Ai agents

Work flow Automation

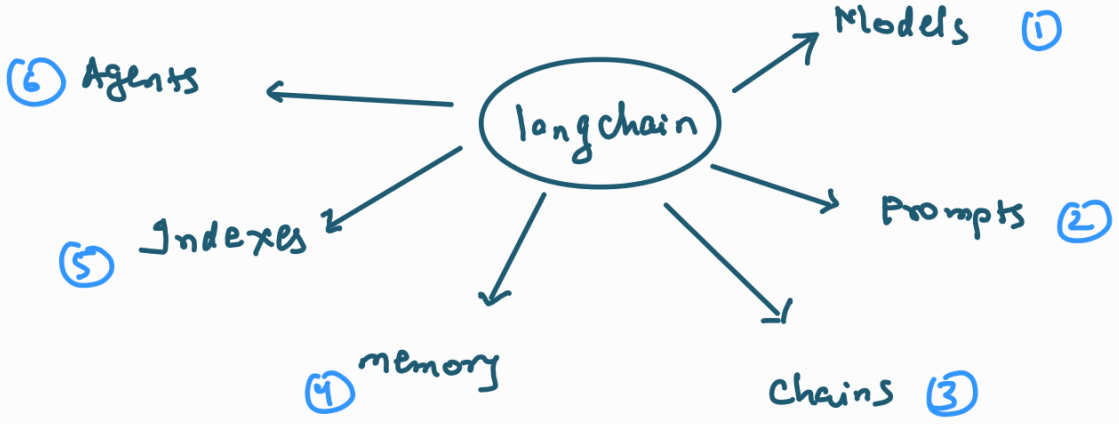
Summarization / research helpers.

Alternatives

i) LLama index

ii) Haystack

LangChain Components :



1) Models:

→ In langchain, "models" are core interfaces through which we interact with AI models

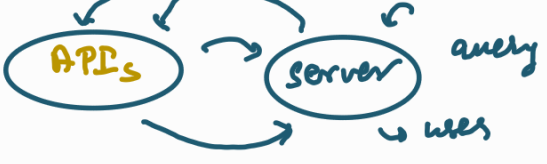
problems before:

i) making the NLP (chatbot) understand (NLU) for eg
eg:- make chat bot understand user query ("check my mail"),

ii) if even chatbot understand, it should provide relavent responses (i.e.) Context aware text generation

These problems: Solved by LLM

NLU
Context Awareness
→ LLMs
trained on
↳ internet
use → one new issue
↓
size (>100 gb)



↓ new issue (implementation)
(every API is implemented differently)

Model
(langchain interface)

eg

OpenAi

①
```
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

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

result = model.invoke("hi who are you?")

print(result.content)
```

Anthropic Ai ↕ (simple standardized model)

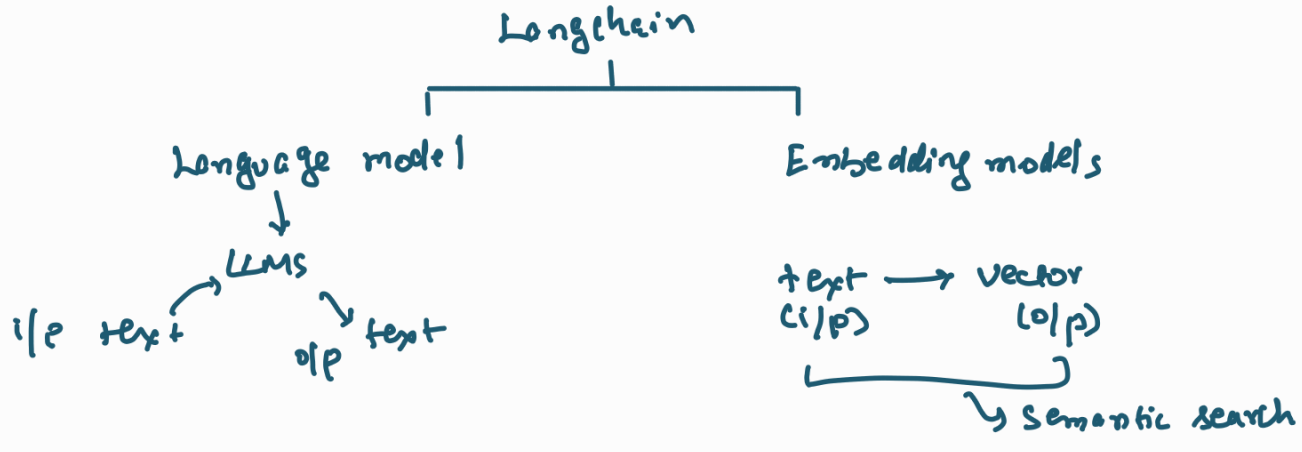
②
```
from langchain_anthropic import ChatAnthropic
from dotenv import load_dotenv

load_dotenv()

model = ChatAnthropic(model='claude-3', temperature=0)

result = model.invoke("hi who are you?")

print(result.content)
```



4) Prompts:

LLM input → prompt → Prompt Engineering → prompt Engineers

(eg: what is an LLM)

i) tone (academic, fun)

i) Dynamic & Reusable Prompts.

```
from langchain_core.prompts import PromptTemplate

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

print(prompt.format(topic='cricket', emotion='fun'))
```

ii) Role-based Prompt

```
chatPrompt = ChatPromptTemplate.from_template([
("system", "hi you are an experienced {profession}"),
("user", "tell me about {topic}"),
])

# format the prompt with variables
formatted_message = chat_prompt.format_message(profession="doctor",
                                               topic="viral fever")
```

iii) few-shot prompt (based on examples)

① examples = [

{"input": "I was charged twice", "output": "billing issue"},

{"input": "The app is crashing", "output": "Technical issue"}

]

② step2 create a template

```
example_template = """
Ticket: {input}
Category: {output}
"""
```

③ step3 : build the few shot prompt template

```
few_shot_prompt = FewShotPromptTemplate(
    examples = examples,
    example_prompt = PromptTemplate(
        input_variables = ["input", "output"],
        template = example_template),
    prefix = "Classify the following customer support tickets into one of the following categories 'billing issue', 'technical issue' or 'general Enquiry'. \n\n",
    suffix = "\n Ticket: {user_input} \n Category:",
    input_variables = ["user_input"]
)
```

---

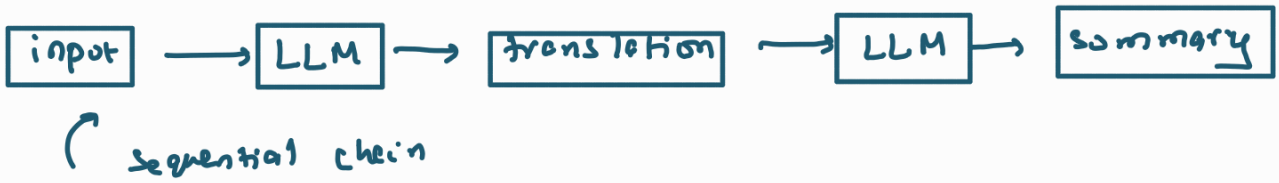
3) Chains

Chains are components using which we can create a pipeline.

eg

input → output

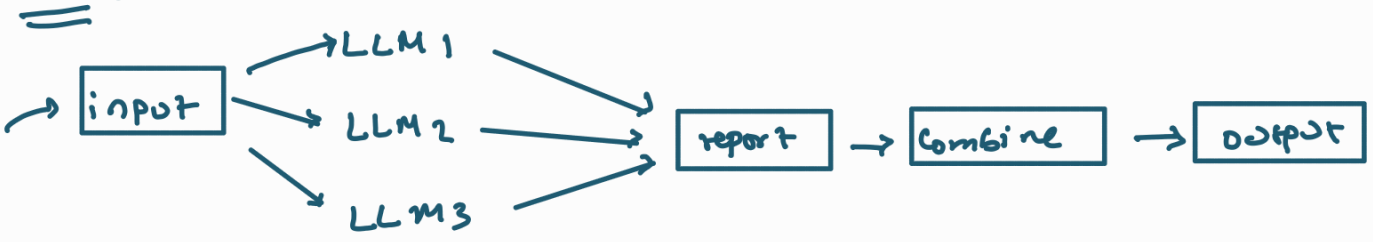
English text (1000) → Hindi summary (100)



Chains → output of one can be given as input to another.

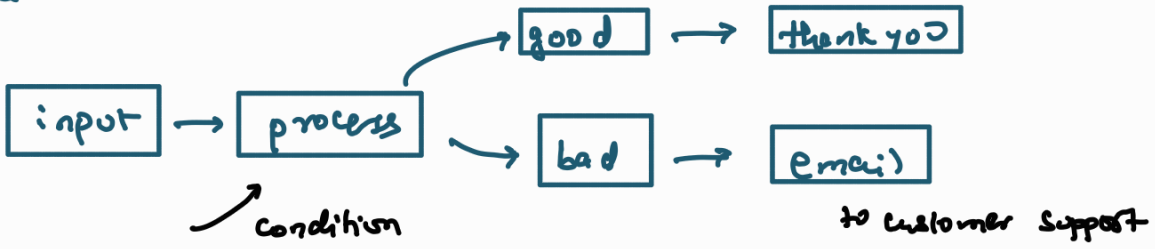
Complex pipelines.

parallel Chain



conditional chaining.

Ai agent feedback



---

4) Indexes

Indexes connect your application to external knowledge such as pdf, websites or databases

Components of Indexes (4)

i) document Loader
ii) Text splitter
iii) Vector store
iv) Retrievers

How?

Load document → Embed document → Store in vector store

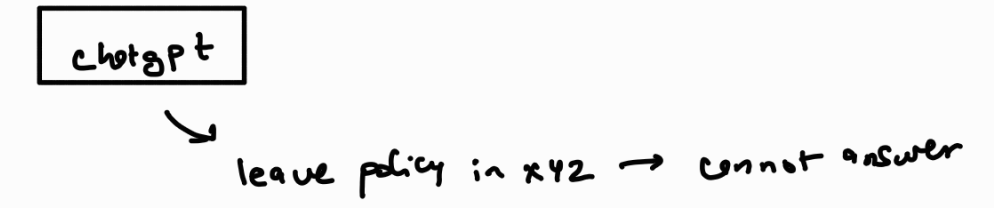
↓

Chain for further processing ← search & retrieve ← Create a retriever

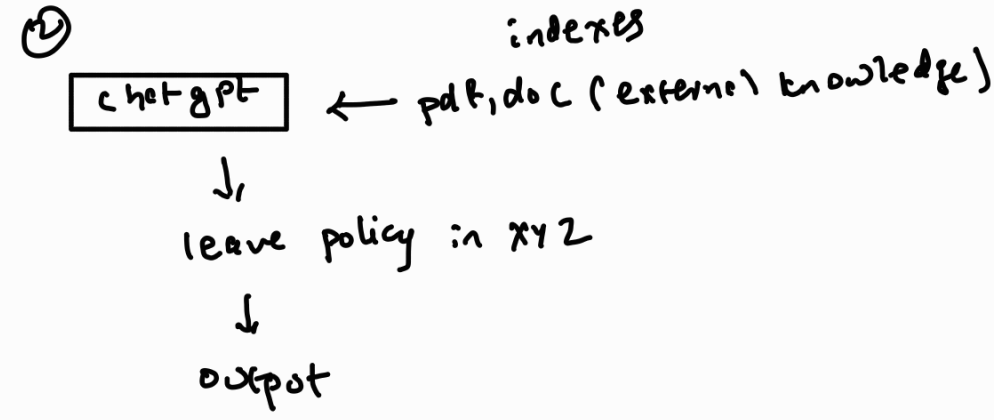
Why indexes?

it cannot answer to questions that require access to private data (we will need to provide it, in the form of pdf, doc (rag) kind of)

①



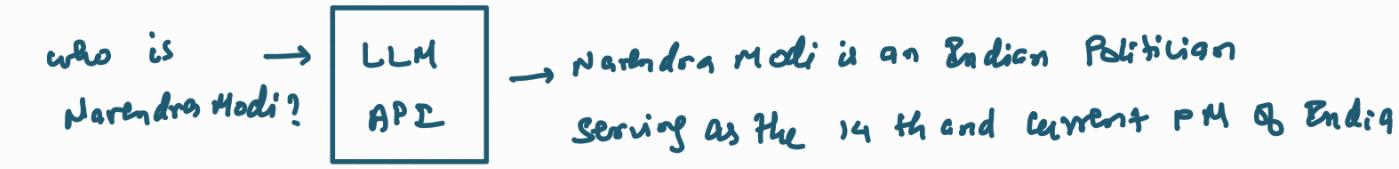
②

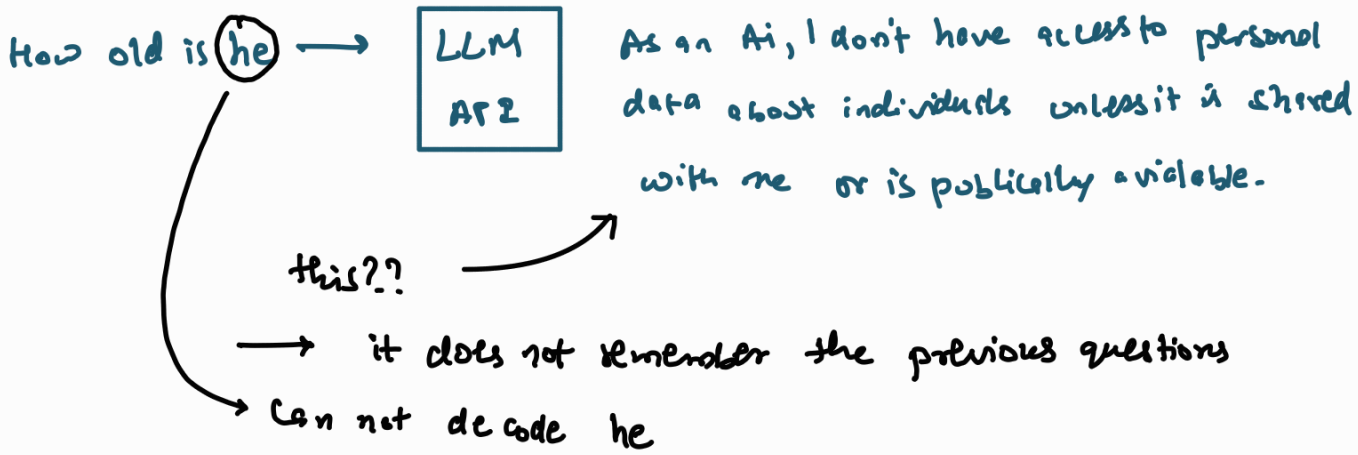


---

⑤ Memory : for storage

LLM API calls are stateless





issues:

i) frustrating chatbot experience (since we need to remind it all the time)

to fix this memory issue, we use this component.

Types of memory:

i) Conversation buffer memory:

→ Stores a transcript of recent messages, great for short chats but can grow large quickly

Chat history is stored and it sends the history of the chat with next Api call

ii) Conversation Buffer Window Memory:

→ Stores last (n) interactions

→ sends only those (n) interactions instead of entire history

iii) Summarizer-Based Memory:

→ Periodically summarizes older chat segments to keep a condensed memory footprint

iv) Custom memory:

→ for advanced use cases, we can store specialized state (eg. the user's performance or

key facts about them) in a custom memory class.

---

6) Ai Agents:

Components designed to perform task autonomously by interacting with external tools or APIs.

↳ LLMs (NLU + context Aware text generation)

↳ Chatbot

eg.

↳ travel website

↳ ai agent (Chatbot + agent)

user:

find cheapest tickets to shimla on 25 Jan

(api call)

user:

book the tickets

(api call)

Ai agent

↳ Reasoning capabilities

↳ access to tools.

tools: calculator, weather API

user: can you multiply today's temperature with 3?

agent: Chain of thoughts

i) to do multiply todays temperature by 3

ii) what is today's temperature → weather api → 25°C

iii) multiply it by 3 → calculator 25 x 3 = 75