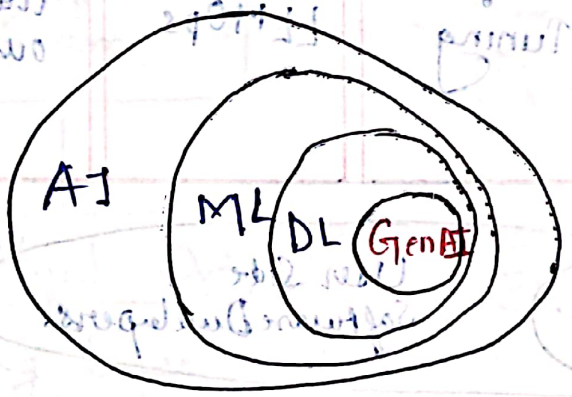
Generative AI

13th August 2025 *
GenAI ! Start from

1. LangChain ✓
2. Langgraph ✓
3. LangSmith ✓
4. Fast API ✓
5. Docker ✓ & Kubernetes
6. AWS ✓ for Deployment
7. Data Science ✓
8. ML Concept & Algo ✓
9. NLP
10. DL (tensorflow, Pytorch)
11. Databases ✓
12. Reinforcement Learning
13. Artificial neural n/w

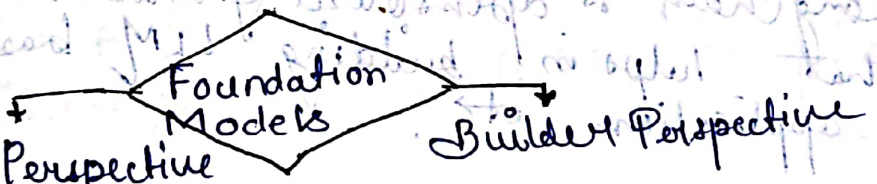
What is ?

Generative AI is a type of AI that creates new content - such as text, images, music or code by learning patterns from existing data, mimicking human creativity !



Working area of genAI –

Customer Support, Content Creation, Education, S/w Development



The Builder's Perspective →

Transfer Architecture

Types of Transformers
- Encoder (BERT)
- Decoder (BERT
- Both based (LT5)

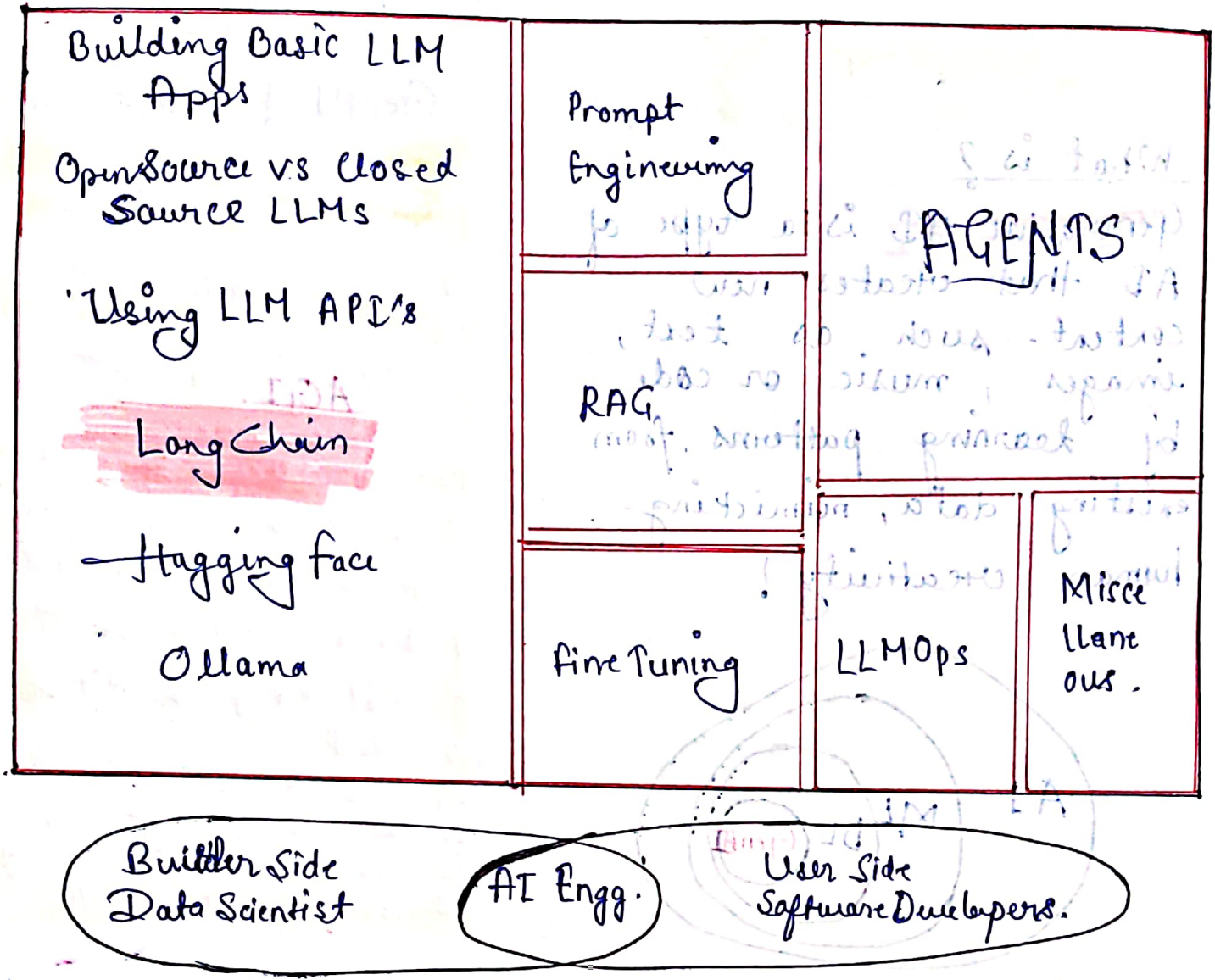
Pretraining
- Training Objectives
- Tokenization Strategies
- Training Strategies
- Handling Challenges

Optimization
- Training Optn
- Model compression
- Optimizing Inference

Fine Tuning
- Task-specific
- Instruction
- Continual Pretraining
- RLHF
- PEFT

Evalution

Deployment



LANGCHAIN → LangChain is opensource framework that helps in building LLM based application etc.

- Fundamentals
- RAG
- Agents.

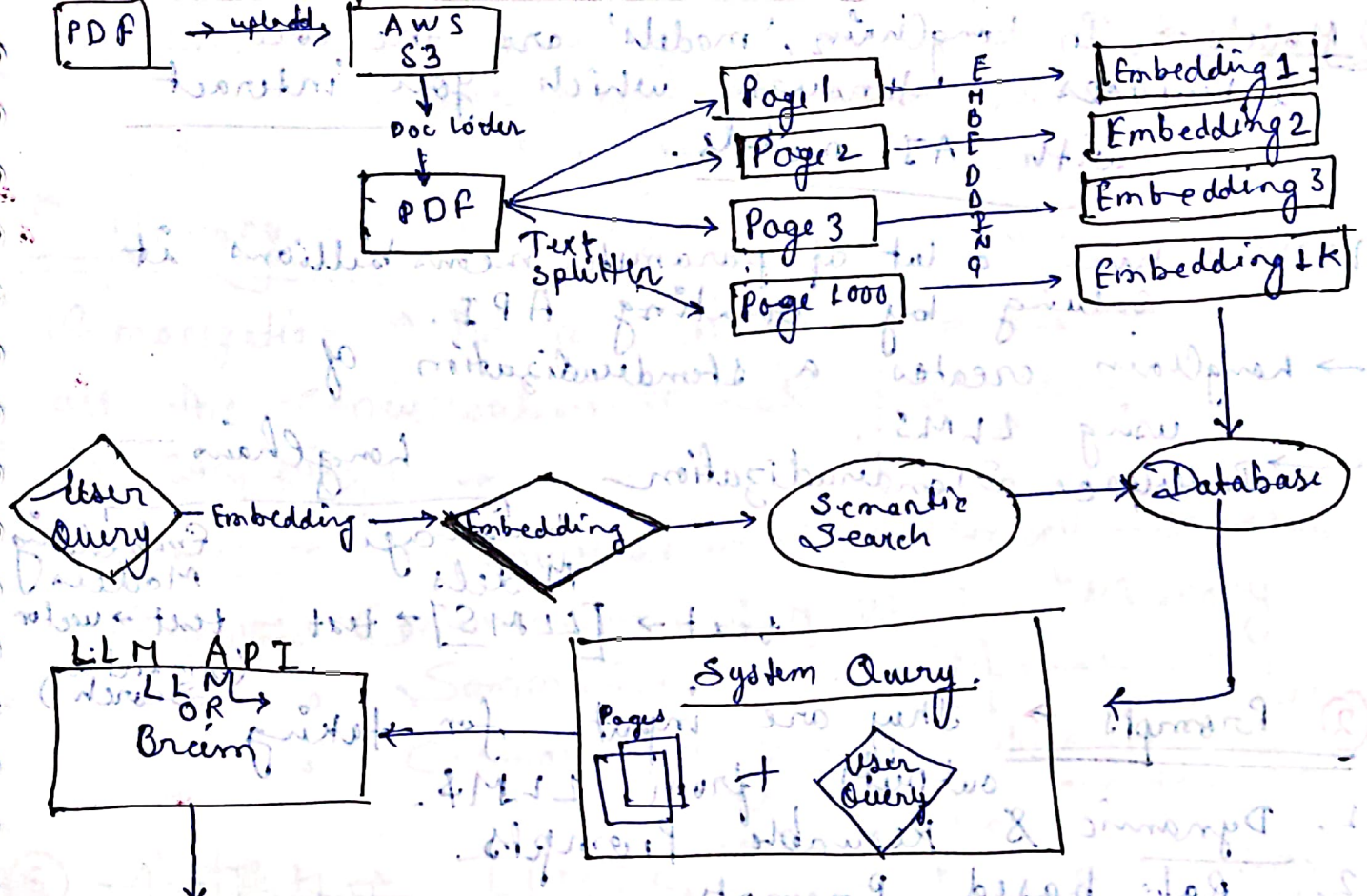
Lecture - 01

NLU - Natural Language Understanding

→ Semantic Search → meaning based searching by embedding (vector) conversion of the input

Alternatives → Haystack, LlamaIndex

→ Context aware text generation means output/reply of the model or system (Ex - Chatbot)



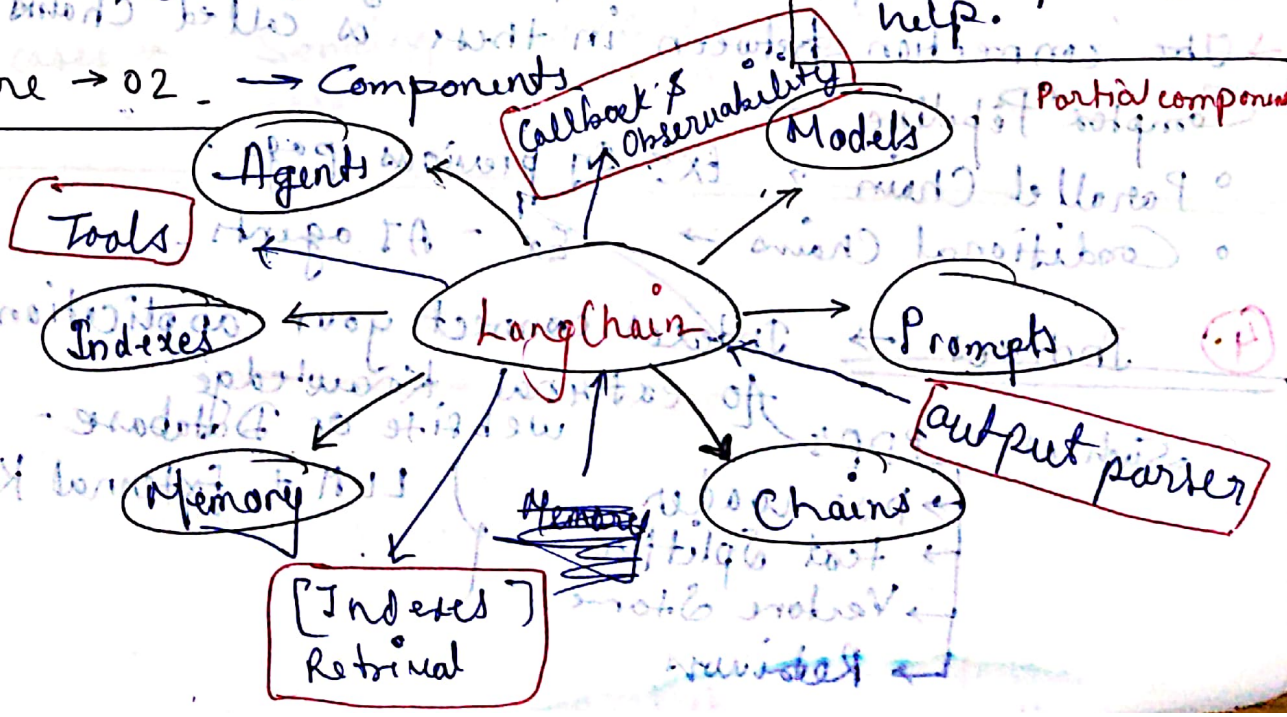
Ex → Structure of PDF Reader using generative AI.

- What we can build..
- Conversational Chatbots
- AI knowledge Assistants
- AI Agents
- Workflow Automation
- Summarise / Research help.

using langchain!

→ Concept of Chains
→ Model Agnostic Development.
→ Complete ecosystem
→ Memory and state handling

Lecture → 02 → Components

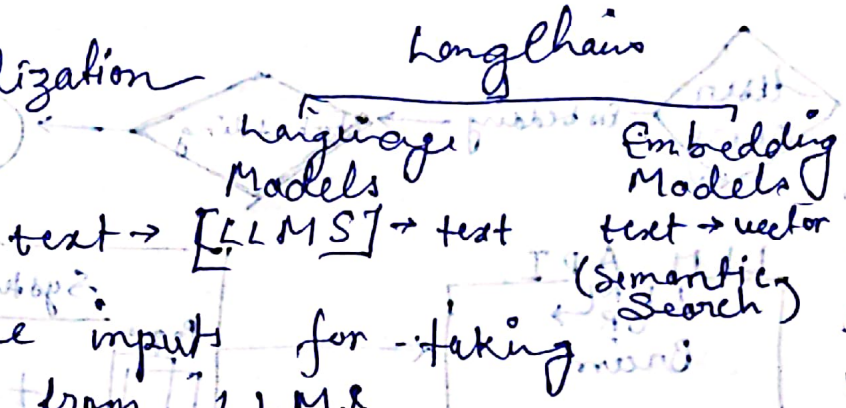


(1) **Models →** In LangChain, 'models' are the core interfaces through which you interact with AI models.

LLMS → have a lot of parameter means billions it saving by creating API.

→ LangChain creates a standardization of using LLMS.

→ Interface Standardization



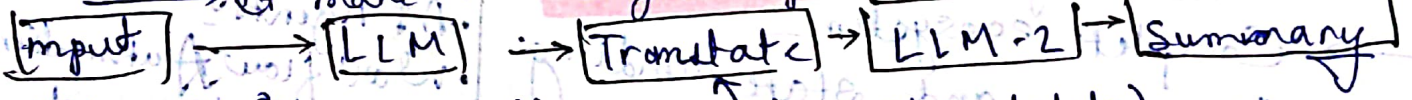
(2) **Prompts →** They are inputs for taking outputs from LLMS.

1. Dynamic & Resuable Prompts
2. Rule based Prompts
3. Few Shot Prompting

(3) **Chains →** For pipeline building LLMS.

Pipeline build is Chains concept → It make one stage output to another stage input.

Some parts by using langchain



→ This is pipeline parts or stage (state)

→ The connection between in these is called Chains

Complex Pipeline →

- Parallel Chain → Ex: In previous page.
- Conditional Chains → Ex - AI agents.

(4) **Indexes →** Indexes connect your application into to external knowledge website or Database.

Such as - PDF's

- Doc loader
- text Splitter
- Vector Store
- Retrivers

LLM + External Knowledge

Indexes → It's the way of building LLMs which has external knowledge source, using which you can build.

⑤ Memory → LLM · API calls are stateless

Memorization → It's a technique to memorize all the conversation or task from starting to stopping! →

Memories →
- Conversation Buffer Memory → Store history
- Conversation Buffer Window Memory → Last n interactions [100]
- Summarizes based - Memory → Store summary of chat
- Custom Memory - Specialised or Specific things store that crutial!

⑥ Agents → LLMS ↔ NLU, text gen. : evolved form of chatbots.
↳ Chatbots + Done some task

It's a AI that can do some real life task for you like your assisstants.

↳ Reasoning Capabilities → They use tools to do task
↳ Tools → It can use API also for task

↳ This work on concept → Chain of thought.

use cases → Sending Mail Automatically also reply them.

Lecture → 03 **Models** 13-08.

The model component in Langchain is crutial part of framework designed to faciliate interactions with various languages models and embedding models.

Models

- **Language Model** → return text
  - LLMs
  - Chatbat
- **Embedding Model** → return vectors [num]
  - Retrival argumented gen (RAG)

Language Model →

They are design AI system to process, generate, and understand natural language text. (NLT)

Free form text generation No built in memory No understanding of roles Ex – GPT-3, Llama 2-27B. LLMs	Optimized for multi conversation Supports structured conversation history GPT-4, GPT.3.5 Turbo, Claude Chatbats.

opensource, closedsource

Huggingface is the largest repo of opensource LLMs

— opensource.

- Using HF Interface API
- Running locally.

14-08-2025

Done practical all kind of model type I had toy.

Temperature → Is the value from 0-2

Is limit that model given output from given input 0-2, 0 means same op altime is going different or creative as increase value but same or deterministic as value decrease

Lecture - 04
2nd Component → **Prompts** - are the inputs instructions or queries given to a model to guide its output.

Prompts
- Text (but now it is use mostly)
- Multimodel prompts ex - Image, video, audio, file.

→ Static — all control in your user hands
→ Dynamic — Prepare a templat so prompt accordingly work

Prompt template → In langchain is a structured way to create prompts dynamically by inserting values into a predefined templates.
instead of hardcoding prompts.

Why →
1. Default validation
2. Reusable
3. Langchain Ecosystem

{ why its use instead over of fstings }

messages → Through an Example.
- System → You are a experienced coder along with 15 years.
- Human → Build UI and logic for langchain chatbot.
- AI → Yes, ofcoure there is professional langchain chatbot by experienced coder with superb UI and a good logic.

→ Chat Prompt templates..

invoke
- Single msg
  - Static
  - Dynamic
- list of msg.
  - Static → System + Human + AI
  - Dynamic

for maintain history
Message Placeholder → It is in langchain is a special type of placeholder used inside a to Chat prompt template. insert chat history or a list of messages at a runtime.

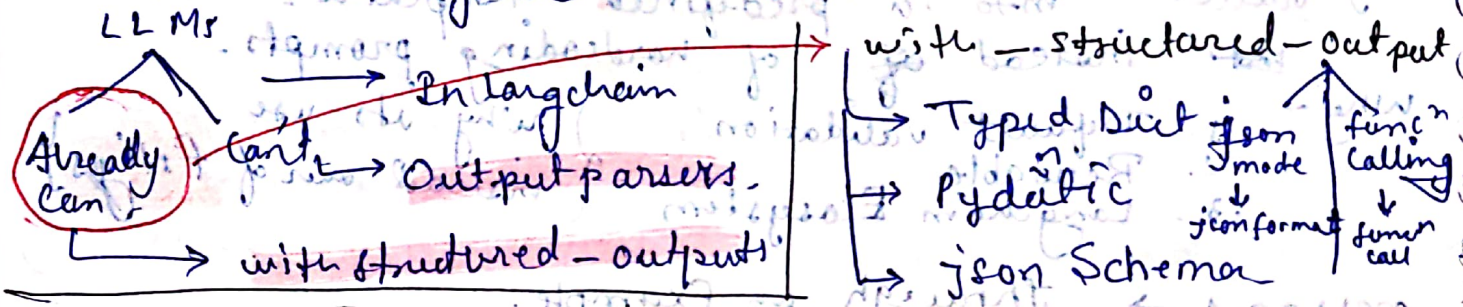
Lecture - 05

Structure Output in LangChain

It refers to having the practice of language models return responses in a well defined data format. (Ex - JSON) rather free-form text. This makes the model O/P easier to parse work with accordingly.

Why! → Data extraction
API Buildings
Agents

It use for connecting LLMs & Chatbots directly to API & Database



- **Typed Dict** → It's a way to define dict in python where you specify what keys and values should exist, helps to ensure that your dict follows a structure.

[Dictionary in Structure called Typed Dict].

→ They define value but validate it.

Annotated → for explaining the LLM about the value output.

Literal → It a restrict variable to specific allowed values. You use literal for give an option of LLM that we want within.

Optional → for choosing O/P or taken for granted

- **Pydantic** → It is a data validations and data parsing library for Python.

It ensure that the data you work with is correct, structured and type-safe

> It require valid data type for we define in class or anywhere
> It can set default value.
> It also have feature Optional fields.
> It also try to understand data by their own and done implicit data conversion as per own (coerce).
> Built in Validation → Ex- Email Str.
> Feild functions → dtype, range (constraint), description
This reduce the hallucinations regex expression.
because the LLm see the schema clearly.

- json_schema → Universal data format

When project have multiple language and variety of data.

Typed Dict	Pydantic	json_schema
→ Only type hints → Dont need validation → You trust the LLL to return correct data.	→ Need Data Validation → Default value if LLM misses field, needed → Automatic type conversion	→ Don't want to import extra python libraries → validation but don't need python object Define structure in in a standard JSON Data format.

15-08

Lecture-06

Structured Output in Langchain – 2

[Built in]

- with_structured_outputs ✓

[Builded]

Can't structure output_parsers by own.

It also work can't and can structure outputs

Output Parser → It help to convert LLM response into structure formate like JSON, CSV, pydantic models and more.

They ensure consistency, validation and ease of use in application.

Output Parsers →

- String Output Parser
- Json O/P Parsers
- Structure O/P Parsers
- Pydantic O/P Parsers.

→ String O/P Parser → The StrOutputParser is the simplest O/P parsers in LangChain. It is used to parse the O/P of LLM and return it as a plain string.

→ Json O/P Parser → It is a quickest way to get json format data from LLM & Chatbots.

→ It doesn't enfore schema. we can't defined format in that.

→ Structured Output Parser → It solve json problem that helps structured JSON data LLM response based on predefined field schema {Response schema}

→ In this we create schema to follow

→ It work by defining a list of fields then model should return a structure output by ensuring a fallow structure.

→ There is no data validation just defining the data.

→ Pydactic O/P Parser → It's a structured o/p parser in LangChain that uses pydantic models.

↳ Why →

- Strict Schema Enforcement
- Type Safety
- Easy Validation
- Seamless Integration

Lecture -07

Chains

→ What
→ Why.

15-08

chain

Prompt → LLM → template → Output

It creates a pipline for the process to working one by one as we defined in a chain.

chain = template | model | parser
then we directly invoke chain
result = chain.invoke({})

output of previous step become a input of next step.

This type chain or pipline is linear or sequential.

Types of Chain

- → Simple Chain ✓
- → Sequential Chain ✓
- → Parallel Chain ✓
- → Conditional Chain ✓

Lecture -08 :

Runables In Langchain :

17-08

In simple term runnable means that functions they invoke func to execute task [generally]

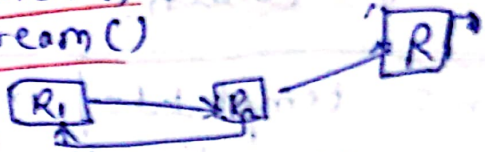
Top used chains :-

1. LLM Chain
2. Sequential Chain
3. Simple Sequential
4. Conversational Retrival
5. Retrival QA Chain
6. Router chain.
7. MultiPrompt Chain
8. Hyder Chain (Hypothetical Document Embeddings)
9. Agent Executor Chain
10. SQL Database Chain

Problem → Too many Chains → Not Seamless connection between Components ✗

So, Runnables → To standardize components.

eg
- Unit of work → input, process, output.
- Common interface → invoke() batch() stream()
- Connection between them



→ a couple of runnable or group of runnable is also a runnable or chain is made.

Lecture - 09

Runnables - 2

→

Task Specific Runnables

→ These are core LangChain that have been converted into Runnables so they can be used in pipelines.

Purpose - Perform task specific like - LLM calls, prompting retrival etc.

Ex - ChatOpenAI
Prompt Template.
Retriver - retrive relevant documents.

Runnable Primitives

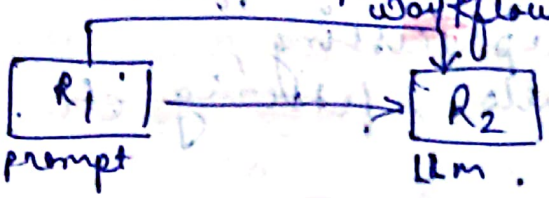
→ These are fundamental building blocks for structuring execution logic in AI workflows

Purpose - They help orchestrate execution by defining how different runnables interact (sequential, conditional, parallel)

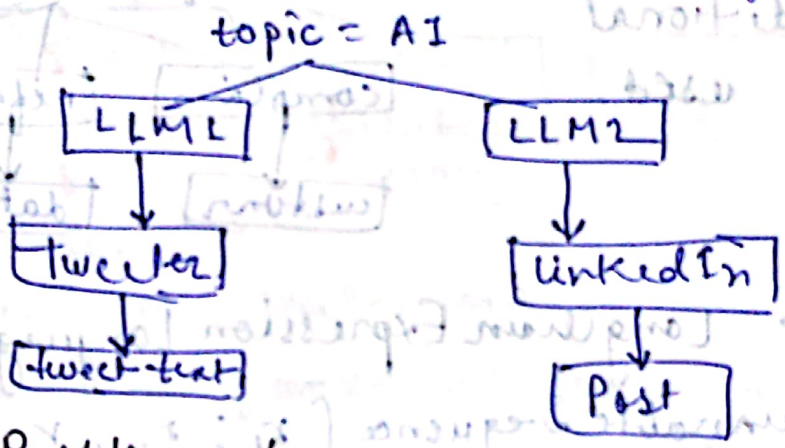
Ex - Runnable Sequence
Runnable Parallel
Runable Map
Runable Branch
Runable Lambda
Runnable Passthrough.

1. Runnable Sequence →

→ It is a chain of runnable in LangChain that execute each after another, passing the output of one step as the input to next → Its useful when you need to compose multiple runnables together in structured workflow



2. Runnable Parallel → that allow multiple runnables to execute in parallel. Each R. receive the same input and processes it independently, producing a dictionary of outputs.



3. Runnable Passthrough →

It is a special type of — that simply returns the input as output without modifying it

Ex — pass = RunnablePassthrough(func^n)

result = pass.invoke("Hello")

Output = Hello.

4. Runable Lambda →

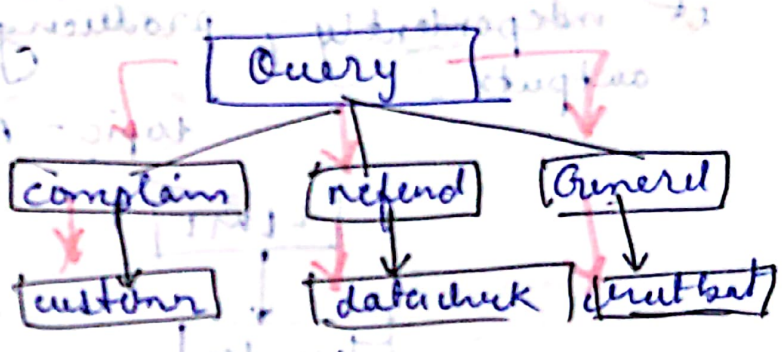
Can Apply python custom functions with in an AI pipeline.

It act as middleware between different AI comp. - enabling preprocessing, transformation, API calls, filtering etc.

5. Runnable Branch →

It is control flow component that allows you to conditionally route input data to different or runnables based on custom logic.

fx -



For conditional logic it used.

LCEL → LangChain Expression Language.

Runnable Sequence [r1, r2, r3]
↳ is declare directly in LCEL

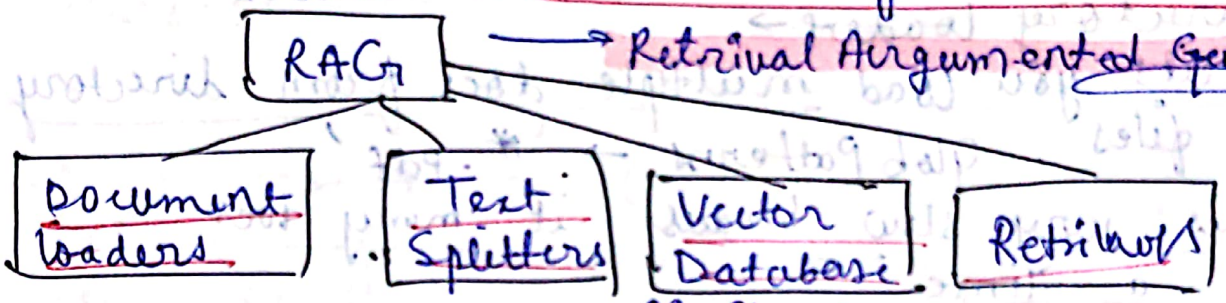
[r1, r2, r3 . . . . , rn]

Lecture - 10
2nd Part.

Document Loaders

RAG is a technique that combines info. retrival ~~that~~ with language generations where a model retrives relevant docs from a knowledge base and then uses them as context, to generate accurate and grounded responses.

1. Use of up to date info
2. Better privacy.
3. No limit of document size.



→ Retrival Argumented Generation

Document Loaders →

Text loader, PyPDF loader, WebBaseloader, CSVloader...

Used to load data from various source into a standaradized format, which can be used for chunking, embedding, retrival and generation.

```
Document (
    page_content = "The actual content of text"
    metadata = {'source': 'filename.txt'} -- }
)
```

1. Text loader → chats, logs, transcripts, code snippets or any plain text data..
   → Works only with .txt files.

18-08-

1 Text loader →

2 PyPDF loader → Simple PDF loader.

It read pdf and convert in document object for each page of PDF

There are issues in multiple scenarios so there are loaders is multiple

table → ○ PDF Plumber loader

Scanned image → ○ Unstructured PDF loader or Amazon Textract PDF loader

○ PyMuPDF loader ← Need image and layout

○ Unstructured PDF loader ← Want best structure in loader.

3 Directory loader →

first lets you load multiple docs from directory or files. Glob Patterns → '*.pdf'

This very slow because it many docs at a time.

→ Load vs Lazy load

Eearger loading	loads on demand.
return - a list of docs	A generator of docs obj.
load all doc immediately in memory.	Docs. are not loaded at once they're fetched one at a time as needed
Best when:	Best when:
The no of docs in less	Your dealing with large docs and a lot
You want exerting upfront.	You want stream processing without lot of memory (eg. chunks, embeddings)

4. Web based loader → use to load and extract text and content from web pages (URLs)

It use Beatyful Soup under the hood to parse HTML and extract visible text.

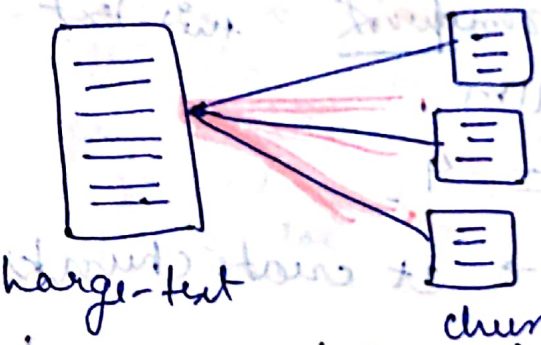
For blogs, news article public website where content is primarily text.	Doesn't JS pages use Selenium URL loader loads only static content

5. CSV loader → load CSV docs in code.

Lecture - 11. 18-08

Text Splitters

It is process of breaking large chunk of text (files; PDF, articles, HTML pages or book) into smaller, managable, pieces that an LLM can handle effectively. {Text-Splitting}



Task	Help
Embedding	Short chunks yield more accurate vector
Semantic Search	Search result point to focused info not noise
Summarisation	Prevents hallucination and topic drift

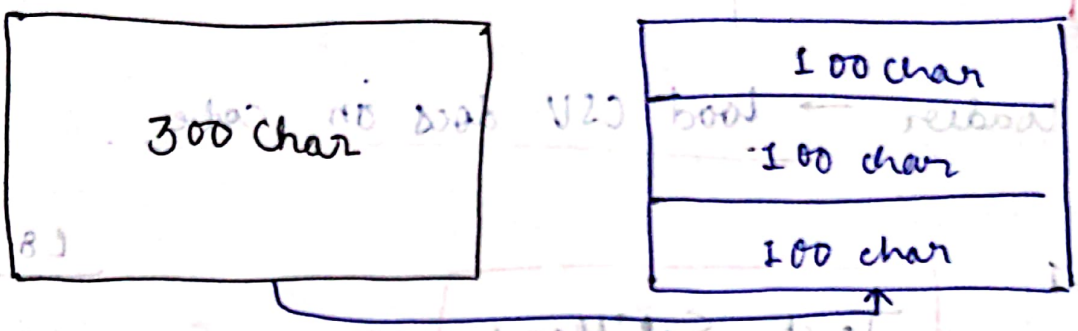
Models have limitations —
So it allows us to process docs that would otherwise exceed these limits.

Optimising Computation resources —
working with smaller chunks of text can be more memory efficient and allow for better parallelization of processing tasks.

Text Splitters

- Length Based
- Text Structure
- Document Structure Based
- Semantic Based search

1. Length Based Text Splitting - eg. 100



→ It just split text not modify or delete/edit it

Chunk_overlap → that text lie in both chunks so the text is overlap in both them -

eg. Langchain is a framework to build llm.. this text is overlapping

chunk - overlap = 4

2. Text Structure Based → It create chunks based on -

para	line	word	characters
\n\n	\n	' '	''

eg. -

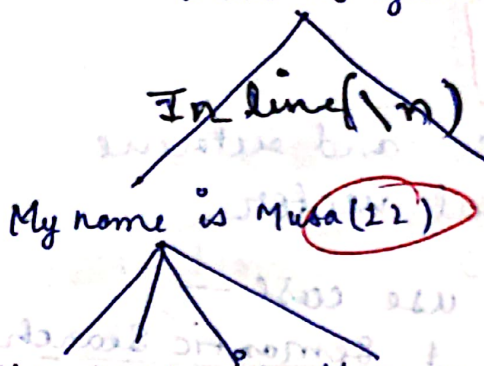
My name is. Musa (12)
I live in Bhopal (12)
I am 19 years old (13)
And I do ing coding (15)

\n\n - para

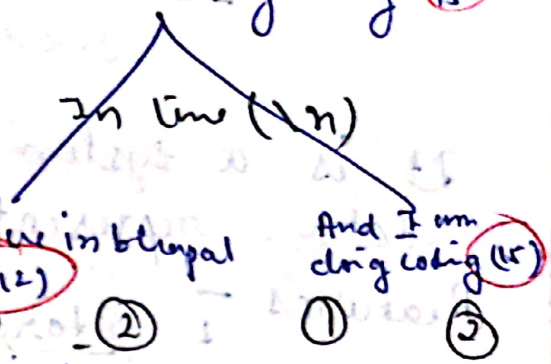
If chunk size = 10

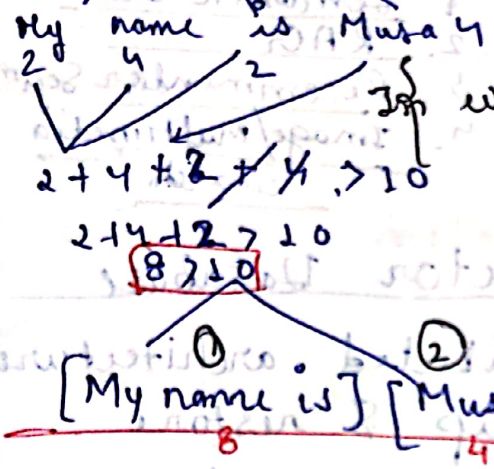
My name is Musa (12) I am 29 years old (13) {In para}

I live in bhopal (12) And I am doing coding (15)



10





In words {' ' or spaces} Same as!

> 10 both under 10 - So total chunk are 8

On words

If chunk_size = 25 50 then chucks are 4 on lines then chunk are 2 - On para for same eg.

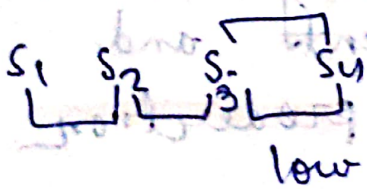
3- Document Structured Based :-

This for special kind of format not for normal text like english or another format.

It is for eg - code snippet, markdown base.

4. Semantic Meaning Based :-

It try to understand while text then it chunk as per their relation this is called meaning based splitter {By using embedding} [for finding similarity in vectors of the text]



Lecture -12

Vector Stores

It is a system design to store and retrieve data represented as numerical vector.

features —
1. Storage memory
2. Similarity Search
3. Indexing
4. CRUD operations

use case —
1. Semantic Search
2. RAG
3. Recommendar System
4. Image/Multimedia Search

Vector Stores	Vector Database
→ Storage System	→ Distributed architecture
→ Sematic Search	→ Backup & restore
→ It is Library.	→ ACID Transaction
Ex - FAISS.	→ Authincation
	→ Concuarrency control
	Ex - Milvus, Qdrant, Weviate

vectorstores + Database feature → Vector Database

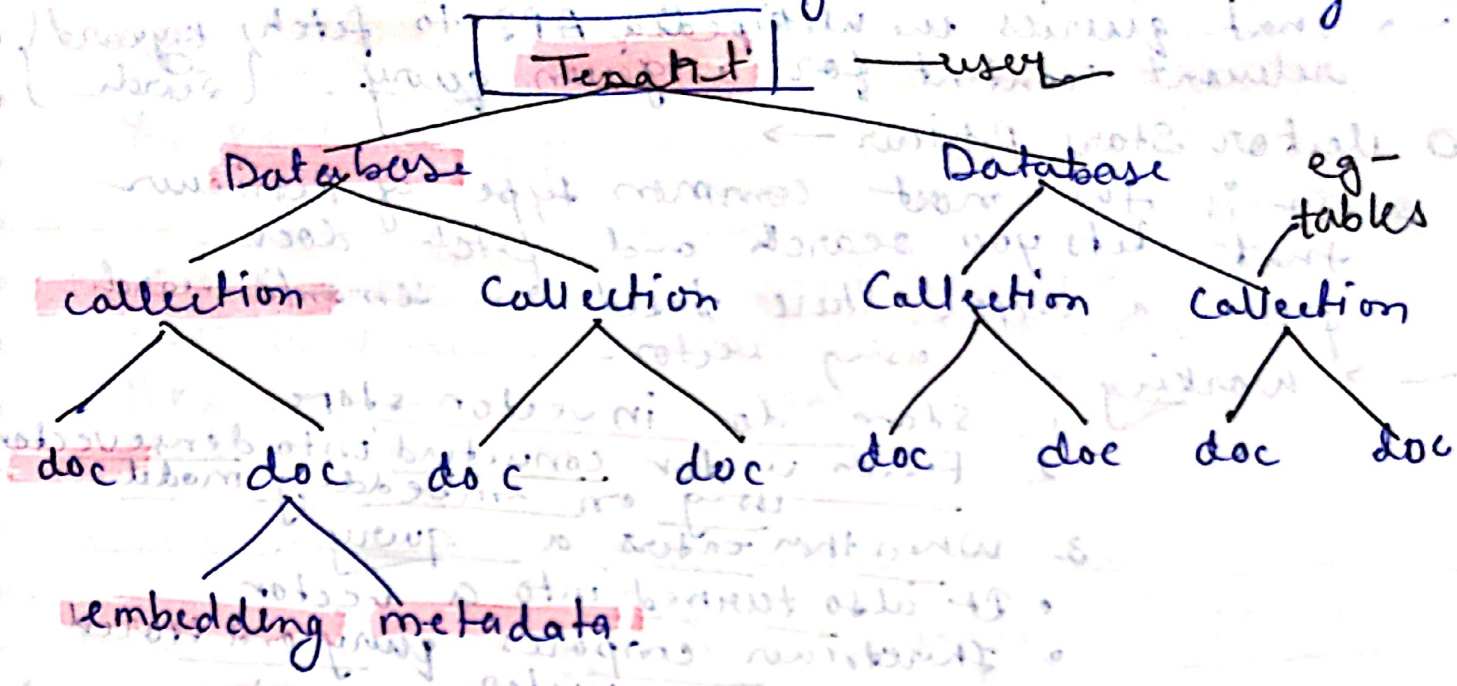
→ It use for clustering, scaling, security metadata, filtering and durability

→ Vector Stores In Longchain —

Chroma Vector Store [Database but lightweight]

→ It is a open source light weight database that is especially friendly for local development and small to medium scale production needs

→ Chroma Tenology and DB Hirachy

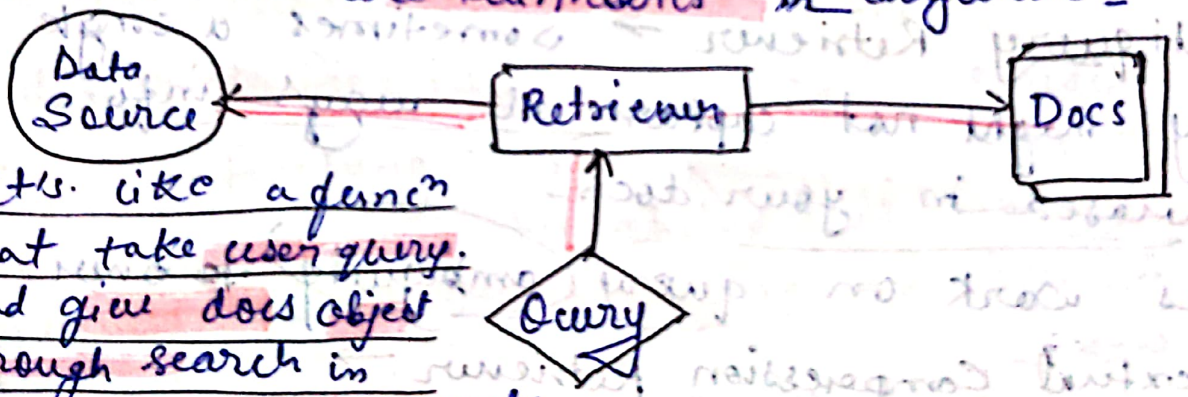


Lecture - 13

Retrievers

A retriver is component in langchain that fetchs relevant docs from a data source in response to users query.

There are multiple type of retrievers →
All retrievers are runnables in langchain -



Its like a func^n that take user query and give docs object through search in data source for relevant results.

Types of →

- Data Source Retrivers
  - wikipedia retriever
  - Vector Space
  - Archive
- Search Strategy Retrivers
  - MMR (Maximum Marginal Relevance)
  - Multi query
  - Contextual Compresion

a. Wikipedia Retriever →
→ that queries the Wikipedia API to fetch relevant content for a given query. {keyword search}

○ Vector Store Retriever →
It is the most common type of Retriever that lets you search and fetch docs from a vector store based on semantic search.

→ Working - using vector -
1. Store doc in vector store.
2. Each vector converted into dense vector using an embedding model
3. When the enters a query.
   - It also turned into a vector
   - The retriver compares query and stored vector
   - It retrives the top k most similar ones

○ Maximal Marginal Relevance →
It is an info. retrieval algo designed to reduce redundancy in the retrived results while maintain high relevance to the query.
pick relevent query but also they are different each other.

○ Multi query Retriever → Sometimes a single query might not capture all ways info. is phrased in your docs.
It is work on query ambiguity to over.

○ Contextual Compression Retriever →
The contextual compression retriver in Langchain is an advanced retriver that improves retrival quality by compressing docs after retrival — Keeping only the relevant content based on user's query.
Remove irrelevant content in docs.

- Your does are long and contain mixed info.
- You want to reduce context length for LLMs
- You need to improve ans accuracy in RAG pipelines.

BM25 Retriver — Parent Document —
Self Query — Time weighted vector —
Arxiv — Ensemble —
Multivector —

Lecture - 14. **RAG** → Theory 18-08.

Query (Prompt) → LLM → Response.

→ Giant transformer based artificial neural n/w based architectures.

They store data in Parameters - It is called Parametric Knowledge

Hallucination ⟶ problems in LLM's. → giving wrong output or ans
- Private dat
- Recent data
- Hallucination

Solution → Fine Tuning ⇒

Supervise finetuning → Train model by labeled data. Eg - Student → LLM, engg → Pretrain, training → finetuning

Continued/Unsupervise finetuning ⇒ Without labelled data -

RLHF → Reinforcement learning + human feedback

→ 1. Collect data - desire prompt - eg.
2. Choose a method - full parameter FT, LoRA/QLoRA or parameter-efficient adapter
3. Train for few epochs -
4. Evaluate & safety test. - exact match, factuality, hallucination rate, red-team for safety

Fine tuning is not feasable —
→ Expensive to do
→ High technical req^n
→ Not work on high change frequency data.

So, there is alternate →

In context learning → It is a core capability of large language models (LLMs) like GPT-3/4, Claude and Llama where the models learn to solve a task purely by seeing examples in the prompt — without updating its weights.

→ It is emergent property of LLM's.

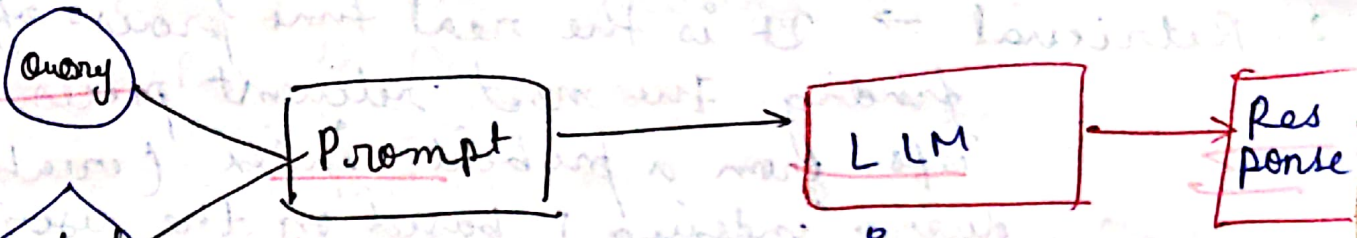
↳ An emergent property is a behavior or ~~daily~~ or ability that suddenly appears in system with it reaches a certain scale or complexity — even though it was not explicitely programmed or expected from the individual components.

→ Instead of just example, retrieve background information, facts, docs, products, manuals etc
Inject that into the prompt to argument the model's knowledge.

→ This is called RAG.

RAG is a way to make a language model [like ChatGPT] smarter by giving it extra information at the time you ask your questions.

Basically personalise, protected, use in given limit and worked only on given env this language model are known as RAG.



Ex - prompt for this →

You are helpful { }
Ans the que only from provided { context }.
If the content is insufficient, just say
don't know.
{ context }
Question : { question }

Understanding RAG → Info retrival
→ text generation

- → Indexing → format data → prepare or create.
- → Retrieval → fetch data
- → Augumentation → Add extra knowledge on parametric
- → Generation → After read prompt → it text gen^n to give o/p.

→ Indexing → It is the preparing your knowledge base so that it can be efficiently searched at query time..

This step consist 4 parts:

1. Document Ingestion → load your source knowledge
2. text Chunking → Break large docs into small semantically meaningful chunks. meaningful chunks
3. Embedding generation → Convert each chunk into a dense vector that captures its meaning
4. Storage in vector store → Store the vectors along with the original chunk text & metadata in vector Database

External Knowledge Base →

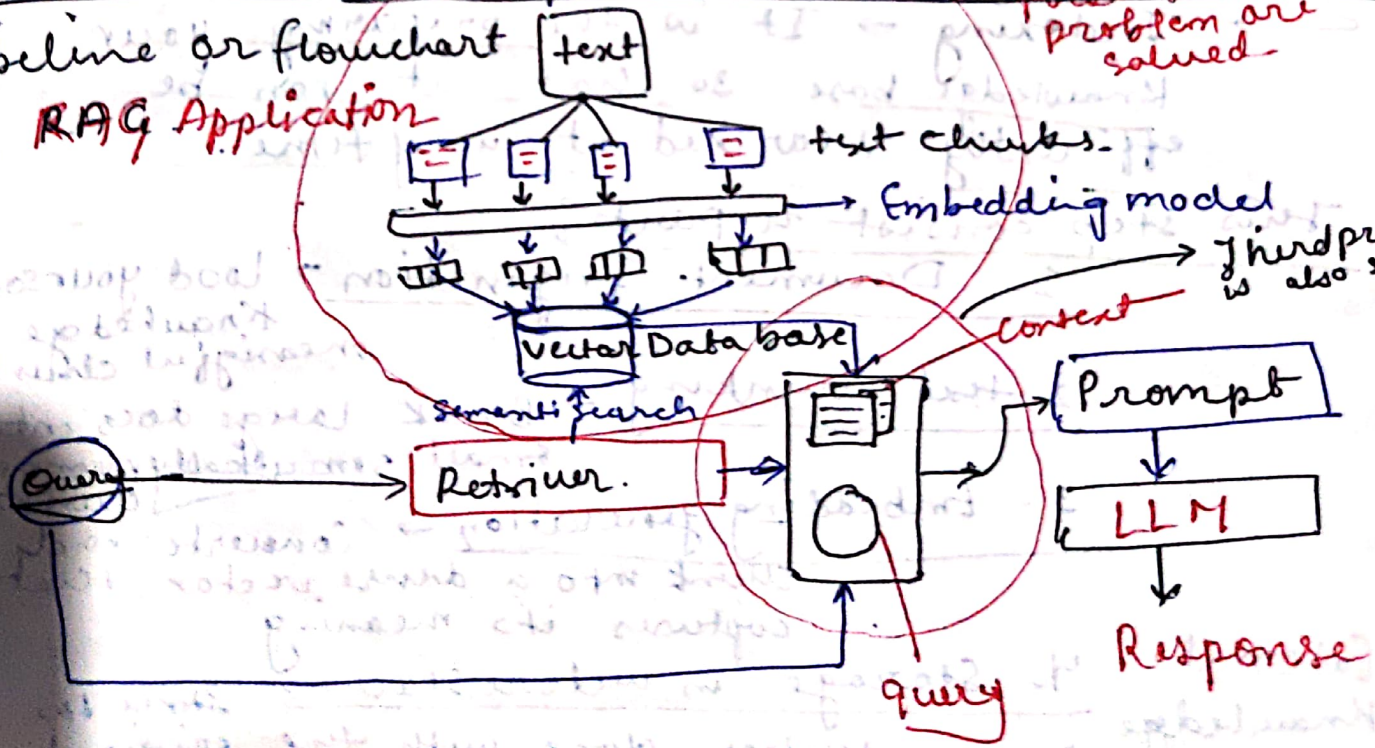
2. Retrieval → It is the real time process of finding the most relevant pieces of info from a prebuilt index (create during indexing) based on the user's questions.

Step1 → Embedding vector of query
Step2 → Searching semantically
Step3 → Ranking { cosine similarity }
Step4 → Most relevent chunk are fetched

3. Argumentation → refers to the step where the retrived docs (chunk of relevant context) are combined with user's query to form a new, enriched prompt for LLM.
It create a [prompt = context + query]

4. Generation → It is the final step where a of LLM, uses the user's query and retrived & argumented context to generate a response.

Pipeline or flowchart of RAG Application



→ It clearly solves problems of LLM -

1. Private Data
2. Recent Data
3. Hallucination.

→ So RAG is optimal alternative of finetuning

1. Cheaper
2. Easier to implement (not complex)
3. Don't require a millions of prepare data
4. Dont need train modal.

Lecture - 15 | RAG - Pratical Done | 19-08

Improvements *

Metrics →
faithfulness
answer_relevancy
context_precicion
context_recall.

1. UI Based enhancements :-
2. Evalution : a. Ragas
   b. Langsmith
3. Indexing :
   a. Docs Ingestion
   b. Text splitting
   c. Vectore Store
4. Retrieval →
   a. Pre-retrival, b. During retrival
   b. Post retrival.
5. Argumentation →
   a. Prompt templating
   b Answer grounding
   c. Context window optimization
6. Generation →
   a. Answer with citation
   b Guard rating
7. System Design →
   a. Multimodel
   b Agentic
   c. Memory Based

Lecture - 16
3rd Part

Tools . In langchain

Create or fetch Tools

19-08.

Tools → Tool Calling → Agents.

Tool → It is just python function (or API) that is packaged in a way the LLM can understand and call when needed.

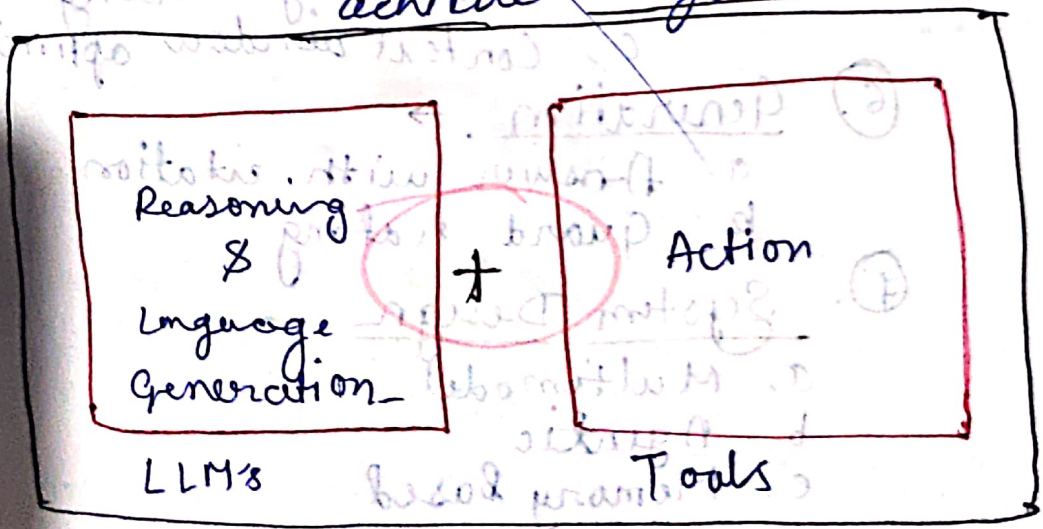
LLMs are great at:
- → Reasoning (Think):
- → Language (text) generation (Speak)

LLMs But they can't
- Access live data
- Do reliable math
- Call APIs
- Run code
- Interact with DB

Tools.
- Built in Tools (general term already in langchain)
- custom Tools (API's call) (Py func^n)

They are also runnables becaus they have invoke func^n

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.



AI - AGENTS.

* Built in Tools →

A built tool is that already langchain provided you

It's prebuilt → DuckDuckGoSearchRun → web search
Wikipedia Query Run → summary
Python REP Tool → Run raw code
Shell Tool → Run shell commands
Request Get Tool → Make HTTP Get req

send email via gmail → Gmail Send Message Tool.

* Custom Tools → user defined tool.

when:

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

1. Create a function
2. Add type hints
3. add tool decorator.

[ways to create tools]

- using @tool decorator
- using Structured tool & pydantic
  - special type of tool where the input to tool follow a structured schema using a pydantic.
- using BaseTool class.
  - It is the abstract base class for all tool in langchain

* It defines the core structure and interface that any tool must follow, wheather it's simple one-liner or fully customized functions.

All other tool types like @ tool, Structured tool are built top of BaseTool.

Toolkits → A toolkit is just a collection of (bundle of) related tool that serve a common purpose package together to form a convenience and reusability *

In LangChain:
A toolkit might be: Google Drive toolkit
And it can contain following tools.

→ Google Drive Create File Tool
Search File Tool
Read File Tool

Lecture - 17

Tools - 2
Calling Tool

Use or Call
Bind
Tools

Tool Binding → Tool binding is the step where you register tool with a LLM so that:
1. The LLM knows what tools are available
2. It knows what each tool does (via description)
3. It knows what input format to use (via schema)

Tool Calling → It is the process where the LLM decides, during a conversation or task that it needs to use a specific tool (func) and generates structured with the name of the tool and the arguments to call it with.

→ The LLM doesn't actually run the tool. It suggest the tool and input arguments. The actual execution done by or handled by langchain or you

Toolkits → A toolkit is just a collection of (bundle of) related tool that serve a common purpose package together to form a convenience and reusability.

In LangChain:
A toolkit might be: Google Drive toolkit
And it can contain following tools.

→ Google Drive Create File Tool
Search File Tool
Read File Tool

Lecture - 17

Tools - 2
Calling Tool

Use or Call
Bind
Tools

Tool Binding → Tool binding is the step where you register tool with a LLM so that:
1. The LLM knows what tools are available
2. It knows what each tool does (via description)
3. It knows what input format to use (via schema)

Tool Calling → It is the process where the LLM decides, during a conversation or task that it needs to use a specific tool (fun°) and generates structured with the name of the tool and the arguments to call it with.

→ The LLM doesn't actually run the tool. It suggest the tool and input arguments. The actual execution done by or handled by langchain or you

Tool Execution → It is the step where actual python function (tool) is run using the input arguments that the LLM suggest during tool calling

Tool Message → It is special type of message it get when executing tool with the help of tool call.

{ currency converter ✓

Lectures - 18

19-08

AI Agents

AI Agent → It is an intelligent system that recive a high level goal from a user and autonomously plans, decides and executes a sequence of action by using external tools APIs & Knowledge source.

All while maintain context resoning over multiple steps adopting to new info and optimizing for the intended outcome.

LLMs + Tools → Agents

Characteristics →

Goal driven

Autonomous planning

Tool using

Context Aware

Adaptive

LLM + Tools → Agent

The process is actually ReAct.

ReAct → It is a design pattern used in AI agents that stands for Reasoning + Acting. It allows a language model (LLM) to interleave internal reasoning (Thought) with external actions (like tool use) in a structured multi step process.

→ Instead of generating ans on go, the model thinks step by step, deciding what it needs to do next and optionally calling tools (APIs, circulator web search etc) to help it.

Useful for →
- Multistep problem
- Tool augmented tasks (web search, DB lookup etc)
  - Thought
  - Action
  - Observation
  → we see that (verbose = True) in agentexecutor
- Making the agent's reasoning transparent and auditable.

AI Agent

Agent

Agent Executor → Thought trace

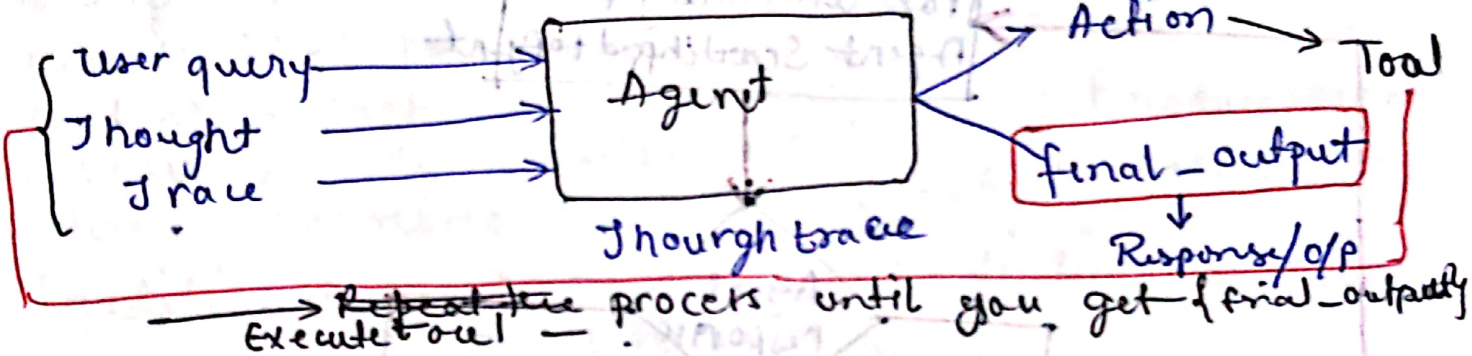
orchestrates the entire loop

1. Sends input and previous message to the agent.
2. Gets the next action from agent
3. Executes that tool with provide input
4. Adds the tool's observation back in the history
5. Loops again with updated history until the agent says final answer.

Creating an Agent →

```
agent = create_react_agent(
    llm = llm,          — Your Model - A1
    tools = [search_tool],  — Tools
    prompt = prompt     — Prefer that from langchain-hub use built in prompt (like purchased 7/react)
)
```

→ Also create Custom prompts. but that is not preferable.



Creating an Agent Executor →

```
agent_executor = AgentExecutor(
    agent = agent,
    tools = [search_tool],
    verbose = True
)
```

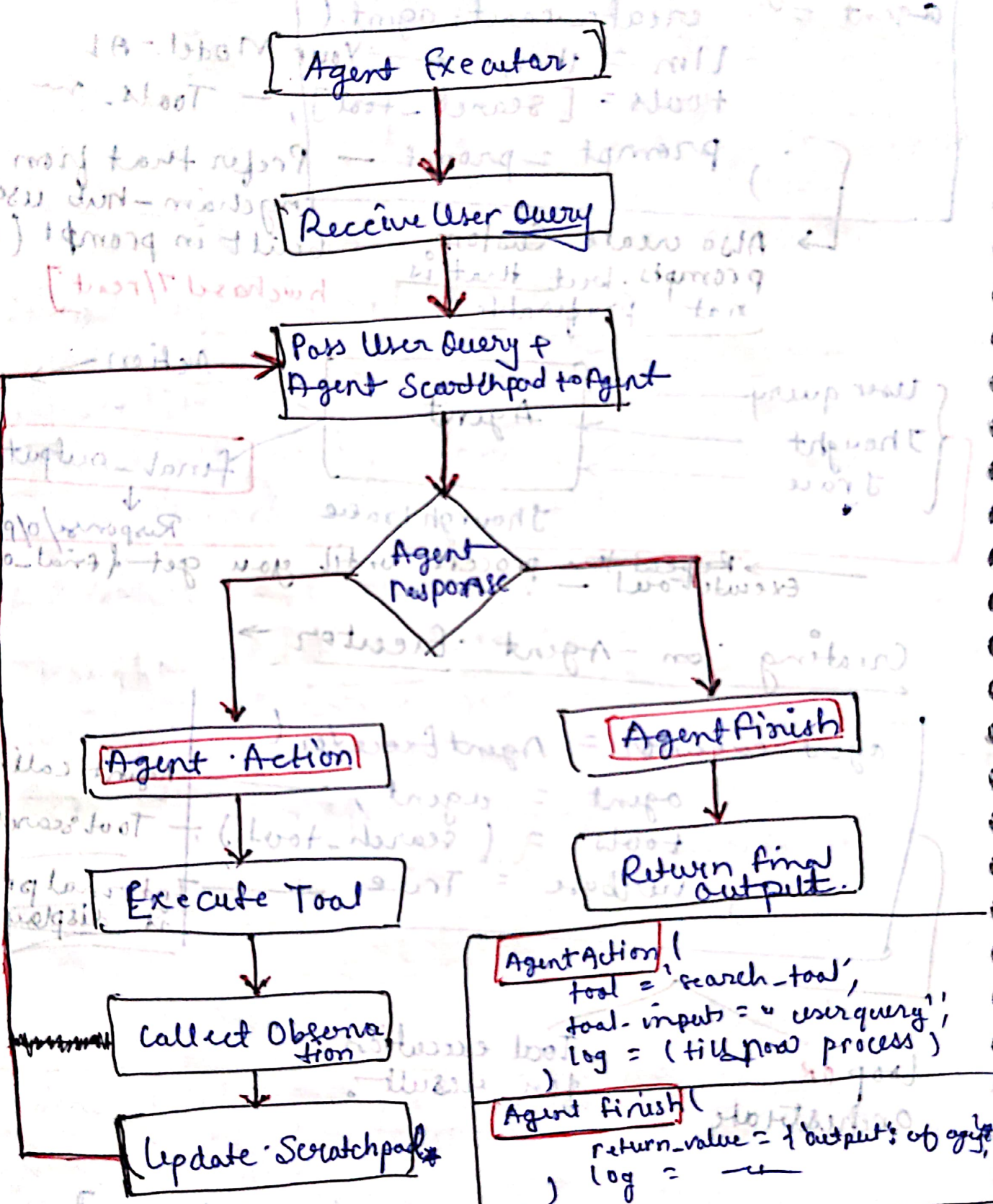
Agent calling
Tool search
Internal process is display.

Loop OR orchestrate

Tool execution for result.

API { Application Programming Interface }

FLOWCHART



WORKING OF AI AGENT.

This langchain library is not efficient for making AI agents instead of use the LangGraph or some other library. (langsmith for monitoring)