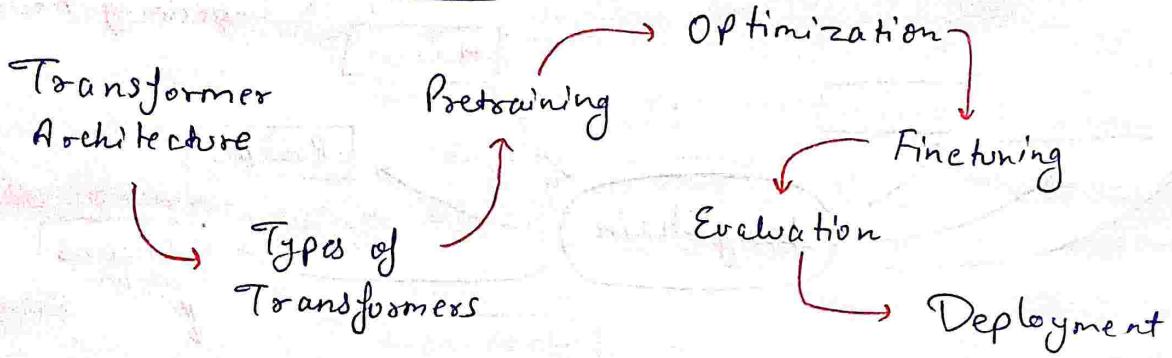
14-04-2026

GEN-AI

Builder's Perspective



User's Perspective

- Building Basic LLM Apps
- Fine tuning
- Prompt Engineering
- RAG
- Agents
- LLMOps

Lang Chain

- open src framework
- supports all major LLMs either open src models or close src
- simplifies developing
- supports many tools & all major GenAI use cases

Benefits

- concepts of chains
- Model Agnostic Development
- Complete Ecosystem
- Memory & state handling

Use cases

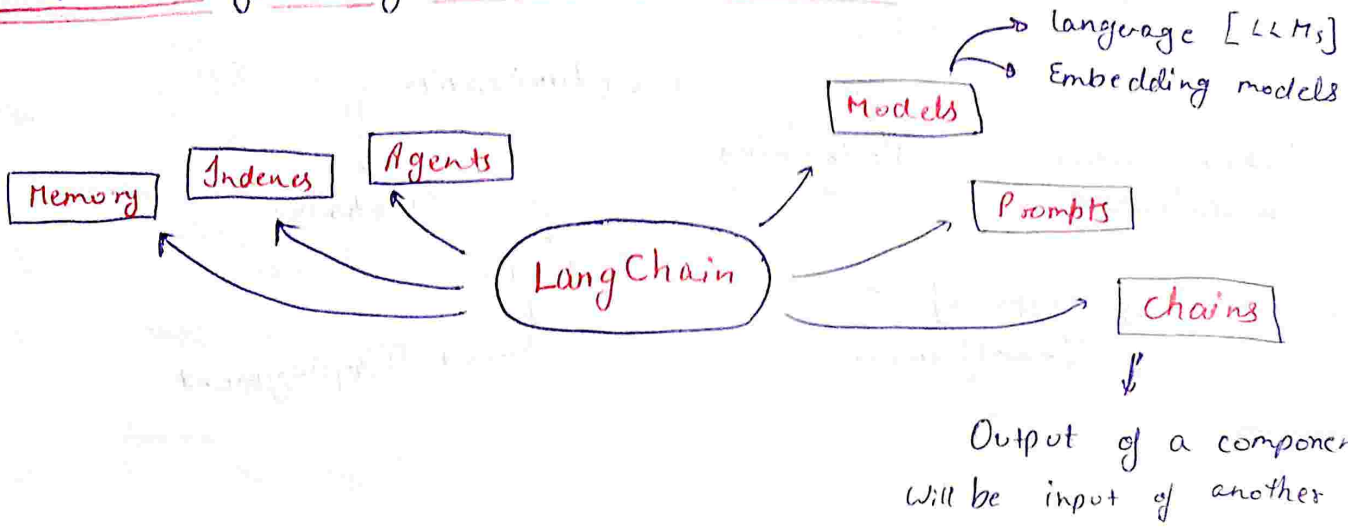
- Conversational Chatbots
- AI Agents
- Workflow Automations

Alternatives of Lang Chain

- Llama Index
- Haystack

RAG: Retrieval Augmented Generation
Ex→ Custom Chatbot on a PDF

Overview of LangChain Components



Indexes

- connect application to external knowledge such as PDF, websites or database

scenario :- If we ask Policy of XYZ company to chatGPT {that chatGPT doesn't train on those policies} than it fails to answer.

main things ↴

- Doc loader
- Text splitter
- Vector store
- Retrievers

LLM + External Knowledge

Memory

LLM API calls are stateless

Q1 → Who is Sunny Deol? → Answer ✓

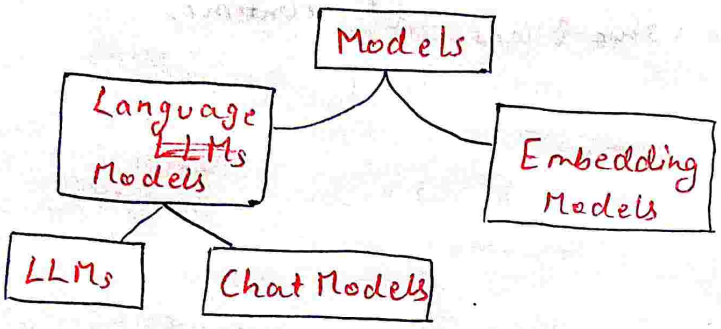
Q2 → How old is he? → Answer ✗

↳ Model/LLM को पता ही नहीं कि हम किसकी बात कर रहें हैं।

- many types, 4 most usable

Models

- Crucial part of framework
- design to facilitate interactions with various language models & embedding models.



Aspect	LLMs	Chat Models
Purpose :	Free-form text generation	Optimized for multi-turn conversations
Training Data	General text {Books, etc}	Fine tuned f. on chat datasets (dialogues, user-assistant conversation
Memory & Content	No built-in memory	Supports structured conversational hist
Role Awareness	No understanding of "user" & "assistant" roles	Understands "system", "user", & "assistant" roles
Examples	GPT-3, Llama-2-7B, OPT-1.3B	GPT-4, GPT-3.5-turbo, Llama-2-Chat
Use Cases	Text generation summarization, translation, creative writing Code generation	Conversational AI, chatbots, virtu assistants, customer support, AI tutors

Prompts

- 2 types
  - Static
    - query → Model
    - → templates → save & load
  - Dynamic
    - query → Template → Model

→ Text Based
→ Multi Modal Input
→ Default validation optional available

Message Placeholder

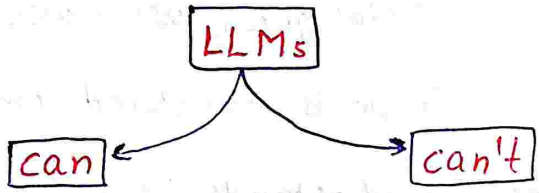
- used inside ChatPromptTem... to dynamically insert chat history or a list of messages at runtime.

Messages

- System Messages
- Human Messages
- AI Messages

] static msg ह but सीधे Dynamic msg बना नहीं सकते "ChatPromptTemplate"

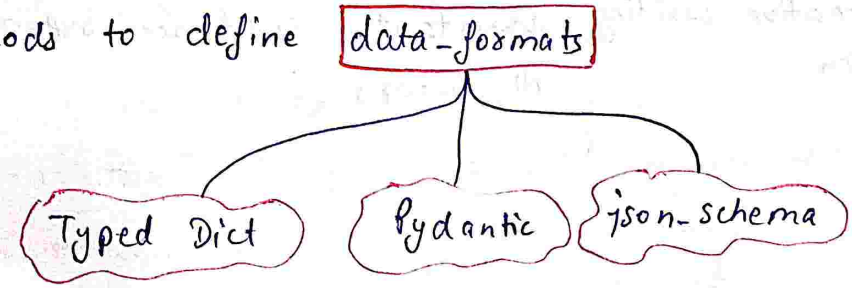
#Structured_output



with_structured_output
(this fn calls before calling invoke,

output_parsers
- we can also use the output parsers with 'can' models

methods to define data_formats



Typed Dict

way to define a dictionary in Python where you specify what keys & values should exist. It ensures that your dict follows a specific structure.

eg: name : str
age : int

अगर age str कर दिया तब भी Typed Dict में error नहीं आता

No-validation

for validation use pydantic

- We can see also sent what about return from LLM by the use of Annotated fn
- We can use pydantic models output as dict & json
- json & typed dict both not have default values feature
- use function calling method in args → method when use OpenAI LLM else json-mode in args → method

Output Parsers

raw LLM response → structured format

String Output Parser → return simple text but useful to making chains
eg → #6 video 20:00

many parsers but we discuss (4)

String opt pars → structured opt par → helps to extract JSON Data from LLM's output
pydantic opt p → JSON opt parser

Cons
↓
No-validation

Json opt parser doesn't enforce a schema

Chains

- Simple
- Sequential
- Parallel
- Conditional

Runnables

- unit of work
- common interface
- easy connection
- runnables ⇒ $R_1 + R_2 + R_3 \cdots$ ⇒ work flow also called Runnable

TYPES

- Task Specific Runnables
  - core langchain components
  - eg ChatOpenAI, ChatPromptTem...
- Runnables Primitive

Runnables Primitive

Runnable Sequence
- simple chain

Runnable Lambda
- covert any python function to Runnable

Runnable Parallel

Runnable Passthrough

Runnable Branch
- control flow component in LangCha that allows you to conditionally route input data to different chains

RAG

- technique that combines information retrival with language generation, where a model retrieves relevant docs from a knowledge base & them uses them as content to generate accurate & grounded response.

Components

→ Document Loaders
→ Text splitters
→ Vector Databases
→ Retrievers

- Use up-to-date information
- better privacy
- No limit of document size

- Text Loader
- PyPDFLoader
- DirLoader x→ DirectoryLoader
  If many docs 100s, 1000s,...
  then use lazy_load()
  ↳ use generator
- WebBaseLoader
- CSVLoader

Document loader ↑.

Text Splitting

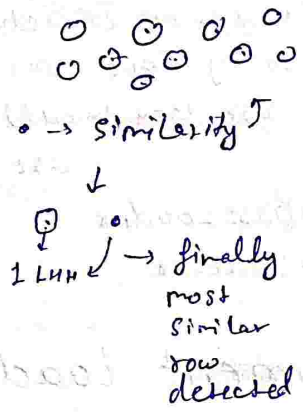
Large Text → Chunks

Splitters

- length based → fast but not more useful → chunks overlap
- text structured " → paragraph, line, word, character
- Document " " → perform by special words ⇒ eg we splitting python code files :-
  class
  def - -
  def - -
  ⋮
  then we specify
  "\ndef", "\n\tdef",
  "\nclass", ...
- Semantic meaning "

Vector Stores

- a system designed to store & retrieve data
- Storage : you can save them
- similarity search
- Indexing : 10 LKH rows → clustering n = 10



- CRUD operations
- eg. Chroma, FAISS, etc

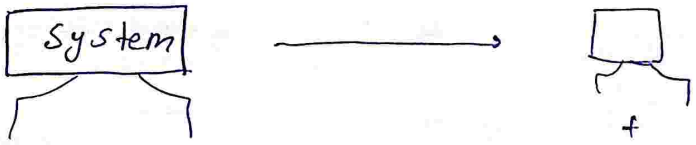
Use-cases

- Semantic search
- RAG
- Recommender search
- Image/Multimedia Search

Chroma Vector Store

- open-src
- small to medium scale production needs
- Lightweight

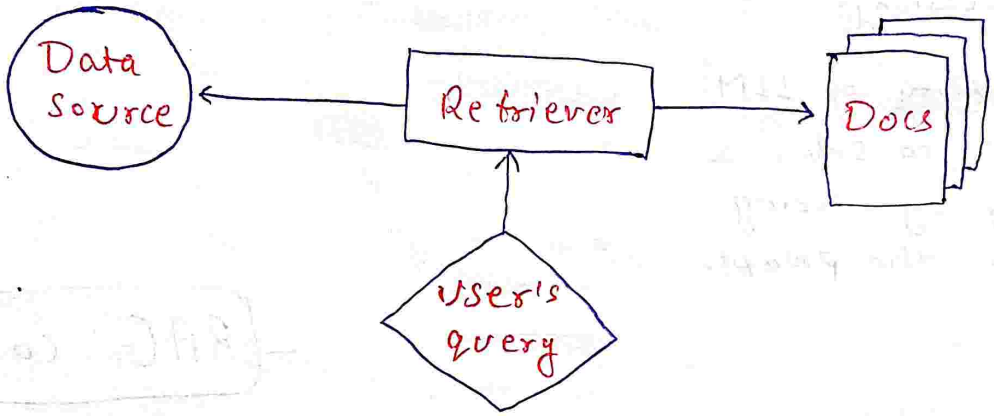
Vector Store V/S Vector Database



- Backup & restore
- ACID transactions
- concurrency, Authentication

Retrievers

- a component in Lang Chain
- It fetches relevant documents from a data source in response to a user's query.
- many types
- All retrievers in the lang Chain are runnables.



- Types
  - wikipedia Ret. } Based on Data
  - Vector store Ret } Based on Data
  - MMR } Based on Search method
  - Multi - Query } Based on Search method
  - etc

Contextual Compression Retriever

- compressing docs after retrieval

query →
document →
imp

MMR

- Maximal Marginal Relevance
- Working
  1st Doc → most relevant
  than rest → most relevant & least similiar
  & so on.

MQR

- Multi Query Retriever
- eg: How can I stay healthy?
could mean
  - How should I eat?
  - How often should I exercise?
  - How can I manage stress?

RAG Concepts

Query → LLM → Response

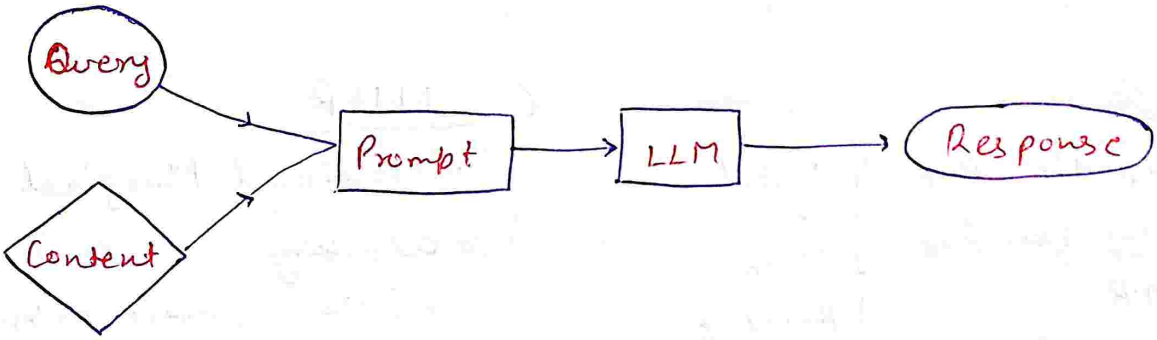
- Problems → sol^{n} : Fine tuning
  - recent data
  - private data
  - Hallucination
  - Problems
    - computation cost
    - Entert technical skills

sol^{n}

In-content learning

- core capability of LLM
- model learns to solve a task purely by seeing examples in the prompt.

RAG: core.c

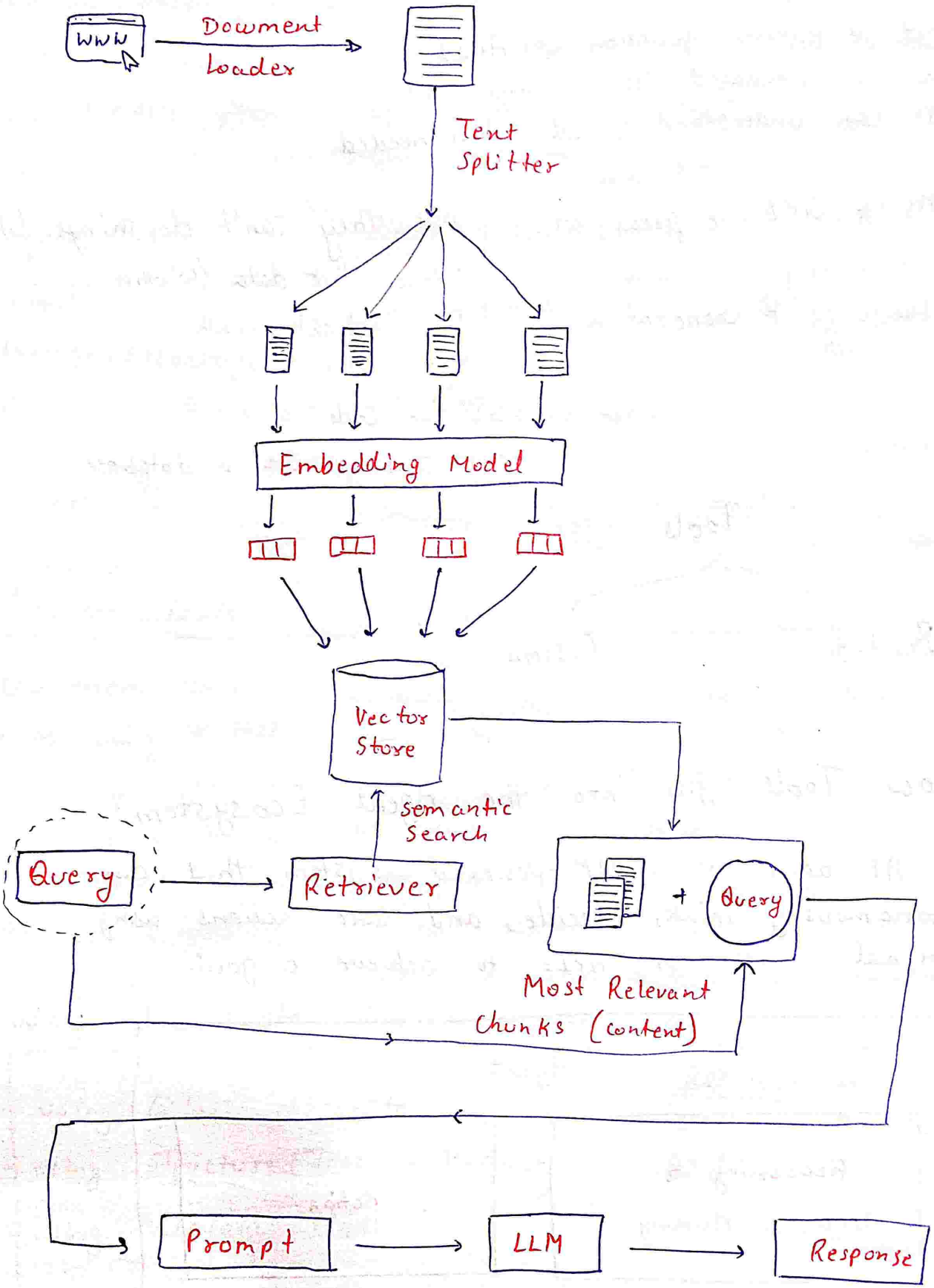


RAG → Information Retrival + Text-Generation

Steps

- Indexing
- Retrival
- Augmentation
- Generation

Basic Architecture

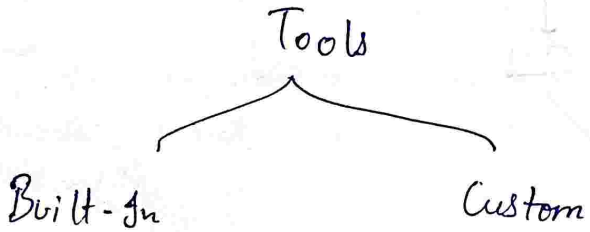


Tools

- just a python function (or API) that is packaged in a way that LLM can understand & call when needed

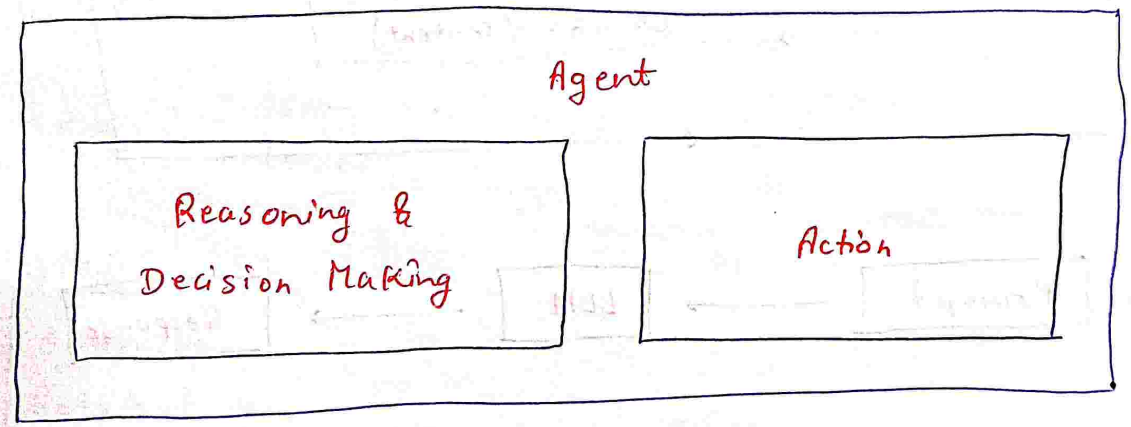
LLMs & GPTs are great at:
- Reasoning
- Language & Generation

But they can't do things like
- Access live data (whether, news)
- Do reliable math
- Call APIs
- Run Code
- Interact with a database



How Tools fits into the Agent Ecosystem?

- An AI-agent is a LLM-powered system that can Autonomously think, decide, and take actions using external tools or APIs to achieve a goal.



Built In Tools

DuckDuckGoSearchRun : Web search via duck duck go

WikipediaQueryRun : : Wikipedia summary

PythonREPLTool : Run Raw Python code

ShellTool : Run shell commands

RequestsGetTool : Make HTTP GET Requests

GmailSendMessageTool : Sends messages via Gmail

SlackSendMessageTool : Post Message to slack

SQLDatabaseQueryTool : Run SQL Queries

:

more

Custom Tools

Use them when:

- You want to call your own APIs
- You want to encapsolate business logic
- You want the LLM to interact with your database, product or app.

Ways to create Custom Tools

- using @tool decorator
- using StructuredTool & Pydantic
- using BaseTool class

BaseTool is the abstract class for all tools in langchain.

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.

Tool Binding

- step where you register tools with a Language Model (LLM) so that:
  - The LLM knows what tools are available
  - It knows what each tool does (via description)
  - It knows what format to use (via schema)

NOTE: All LLMs does not support Tool binding

Tool Calling

It is the process where the LLM decides, during a conversation or task, that it needs to use a specific tool (function) — and generates a structured output with:

- the name of the tool
- the arguments to call it with

NOTE: LLM does not actually run the tool - it just suggests the tool and the input arguments, The actual execution is handled by LangChain or you

Example

query → what's 8 multiplied by 7?

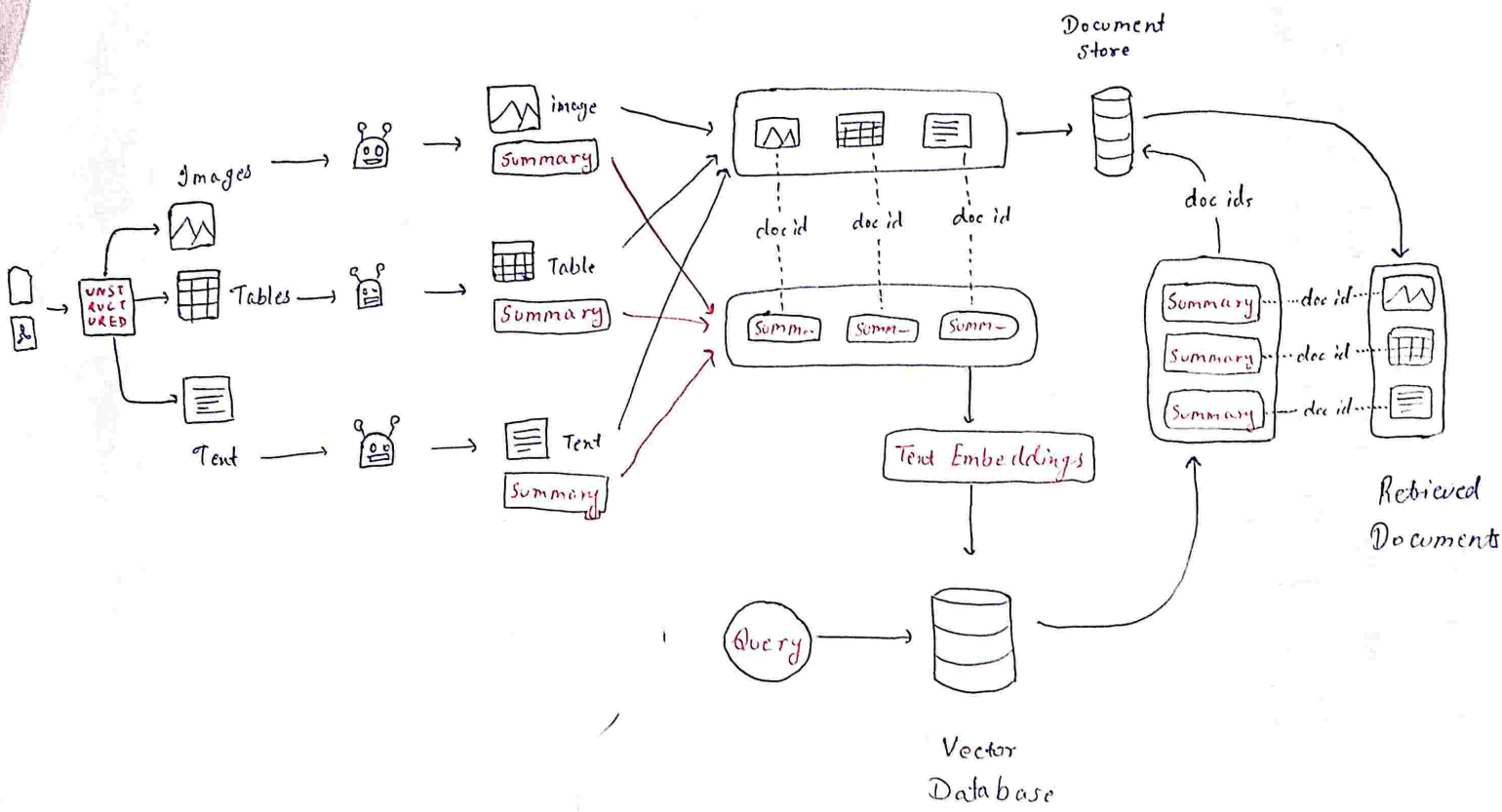
The LLM responds with a tool call:

```
{ "tool" : "multiply",
  "args" : { "a" : 8, "b" ; 7 }
}
```

Tool Execution

It is the step where the actual Python function (tool) is run using the input arguments thats the LLM suggested during tool calling

Multimodal Retrieval using Unstructured For Extraction



More Topics

- ✯ Evaluation ⟶ Metrics
  - faithfulness
  - answer relevancy
  - content Precision
  - content Recall
  - Rags
  - LangSmith
- ✯ Indexing
  - Document Ingestion
  - Text splitting
  - Vector Store
- ✯ Retrieval
  - a) Pre-Retrieval
    - Query Rewritting using LLM
    - Multi-Query generation
    - Domain aware routing
  - b) During Retrieval
    - MMR
    - Hybrid Retrieval
    - Reranking
  - c) Post Retrieval
    - Contextual Compression
- ✯ Augmentation
  - Prompt Templating
  - Answer grounding
  - Contextual Window Optimization
- ✯ Generation
  - Answer with Citation
  - Guard Railing
- ✯ System Design
  - Multimodal
  - Agentic
  - Memory based