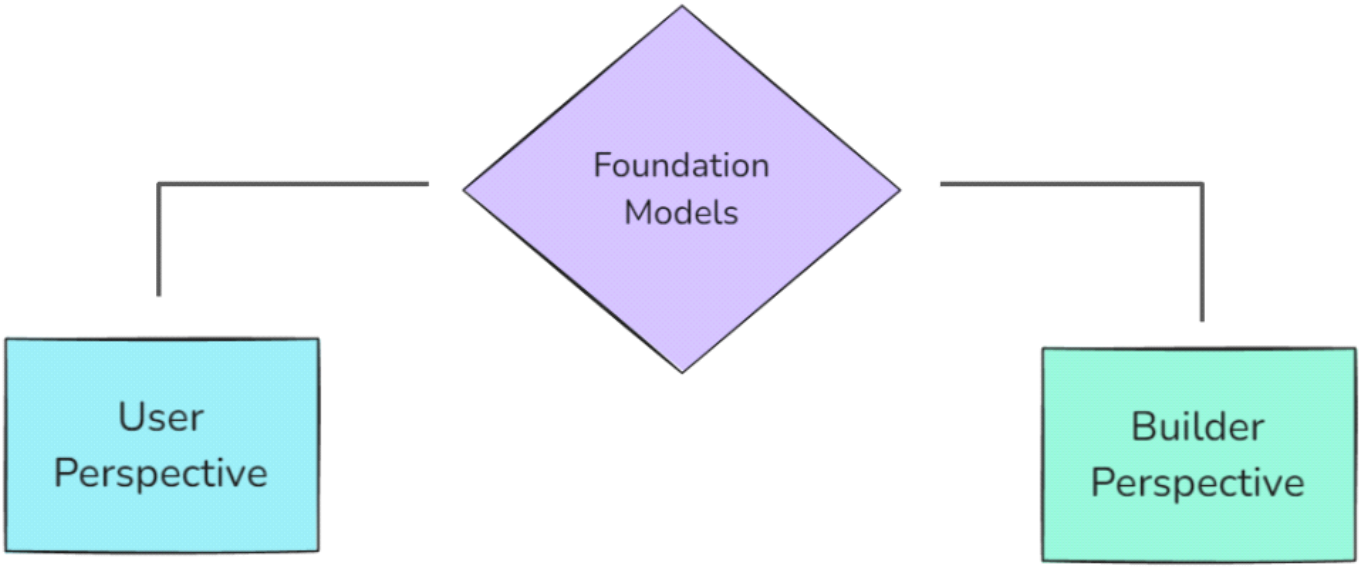
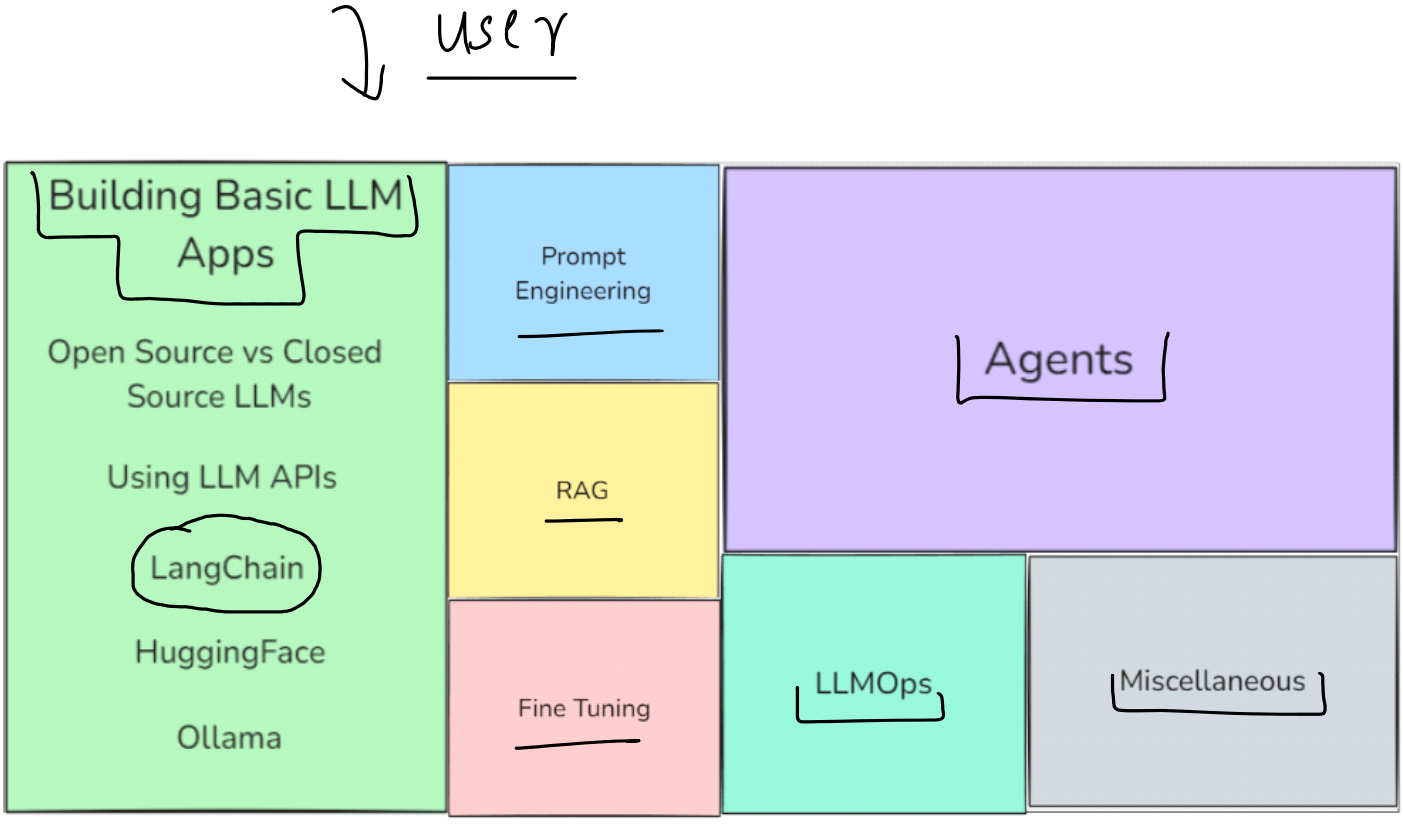
A little background!

29 January 2025 08:28





What is LangChain

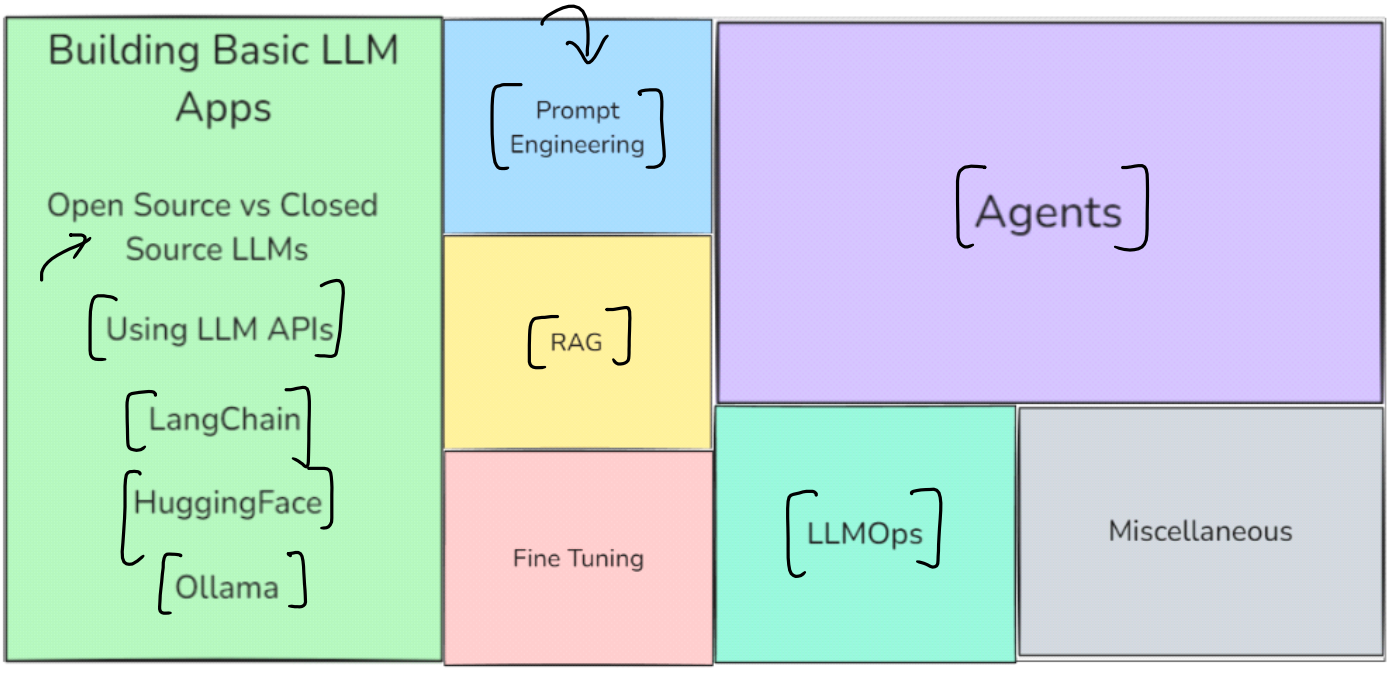
29 January 2025 08:35

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.

1. Supports all the major LLMs
2. Simplifies developing LLM based applications → chains
3. Integrations available for all major tools
4. Open source/Free/Actively developed
5. Supports all major GenAI use cases

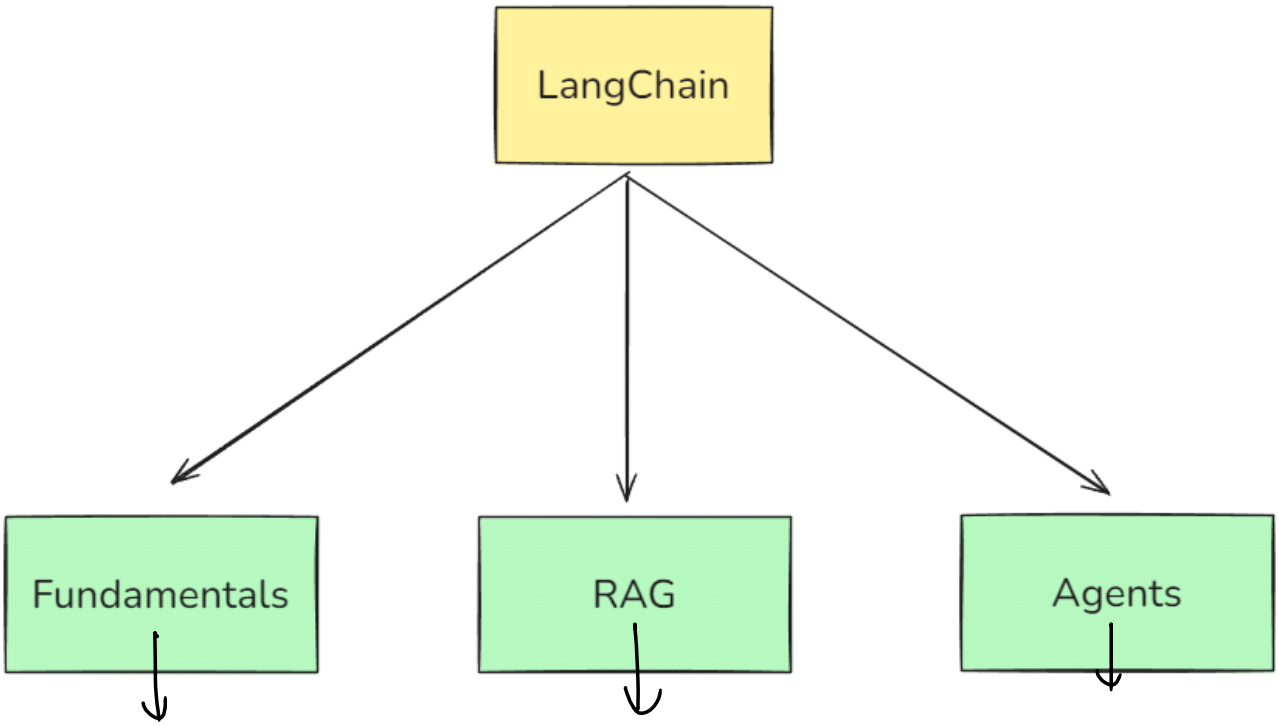
Why LangChain first

29 January 2025 08:35



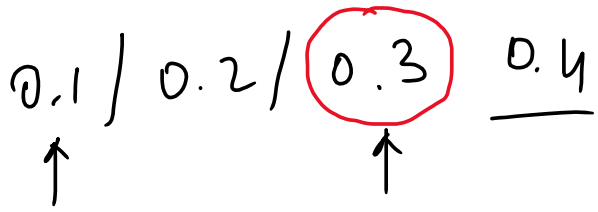
Curriculum Structure

29 January 2025 08:36



My Focus

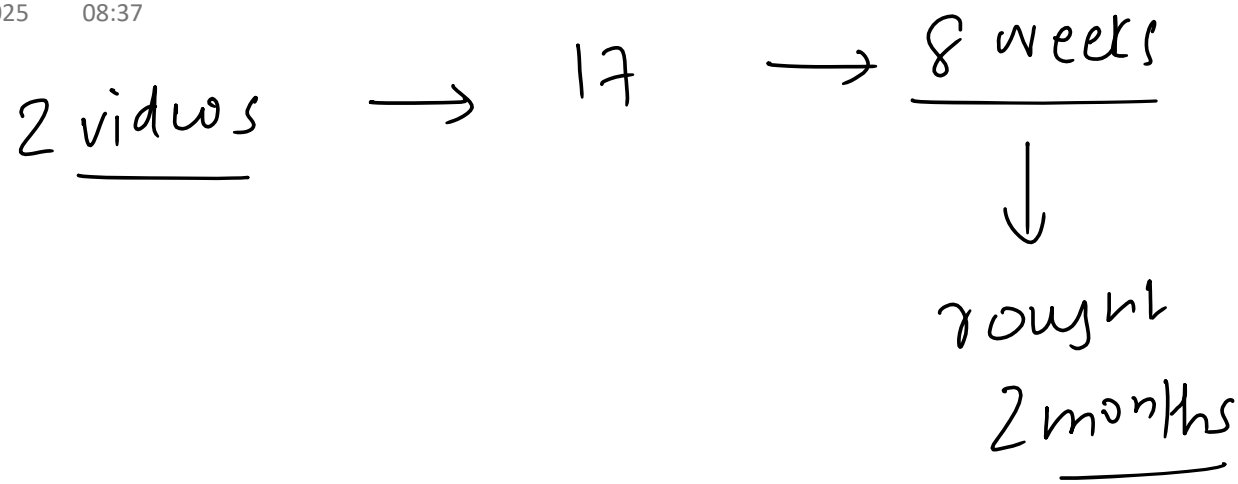
29 January 2025 08:35



1. Updated information
2. Clarity
3. Conceptual understanding
4. The 80 percent approach

Timeline

29 January 2025 08:37



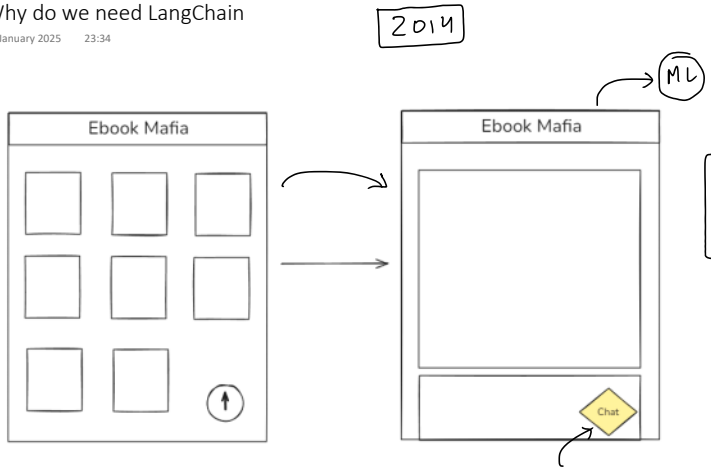
What is LangChain

07 January 2025 23:14

LangChain is an open-source framework for developing applications powered by large language models (LLMs).

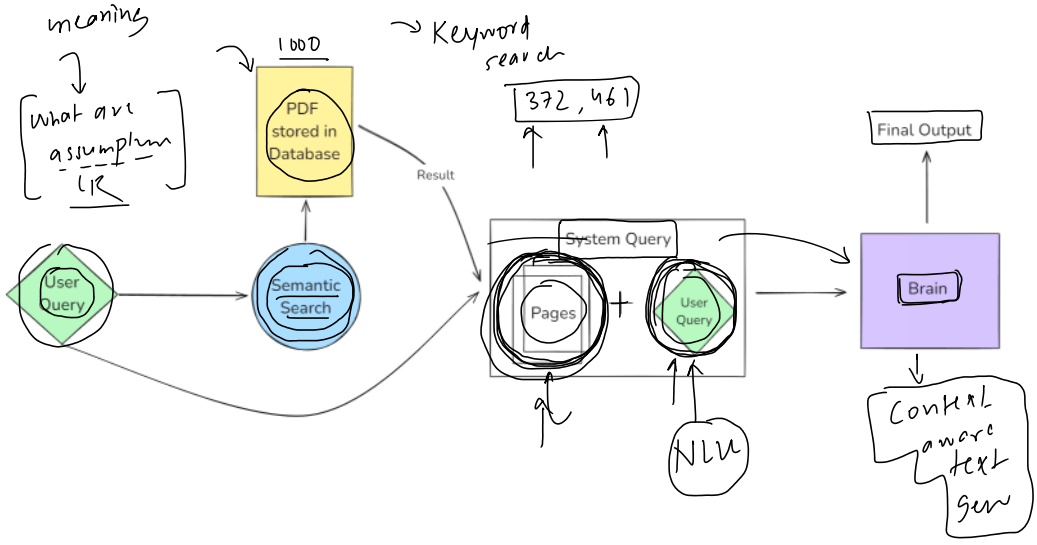
Why do we need LangChain

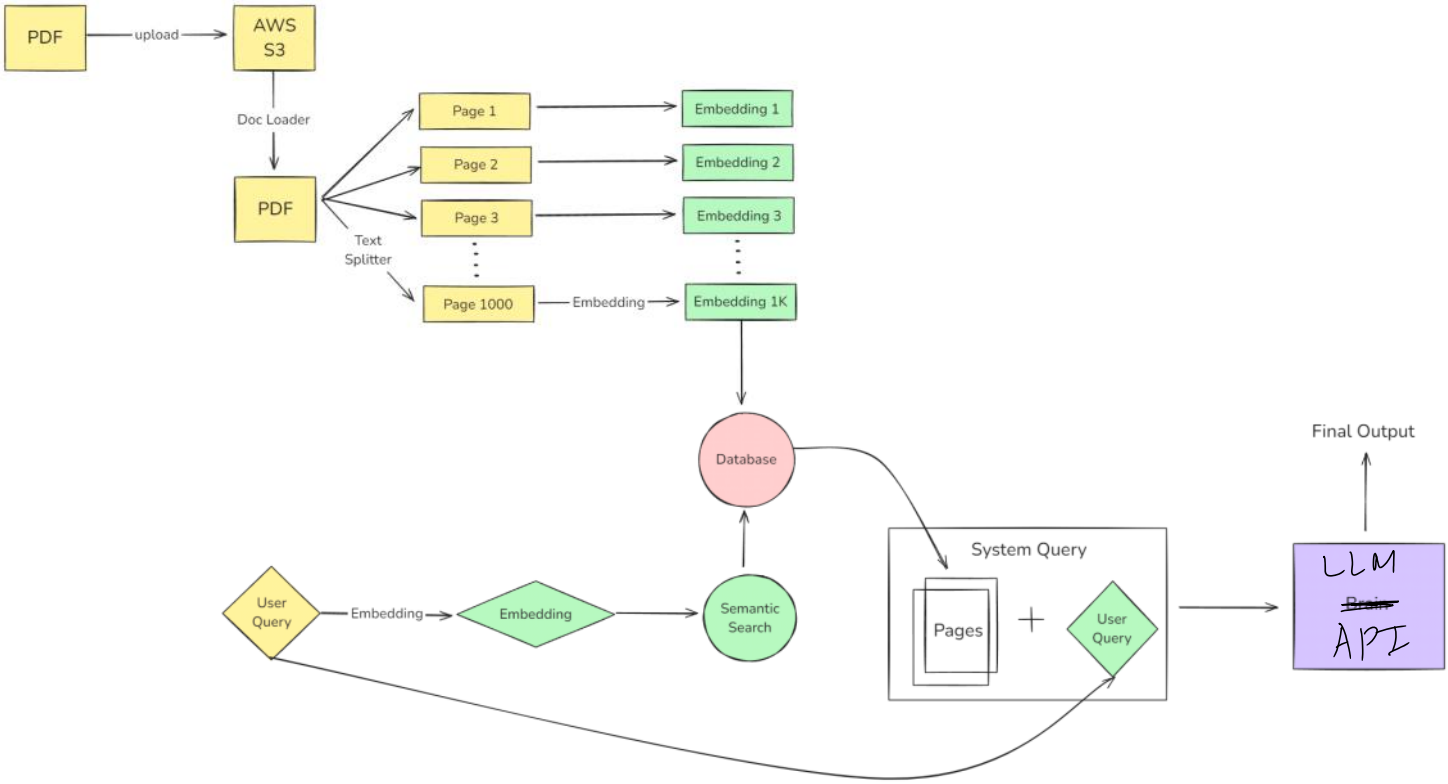
21 January 2025 23:34



Example Queries

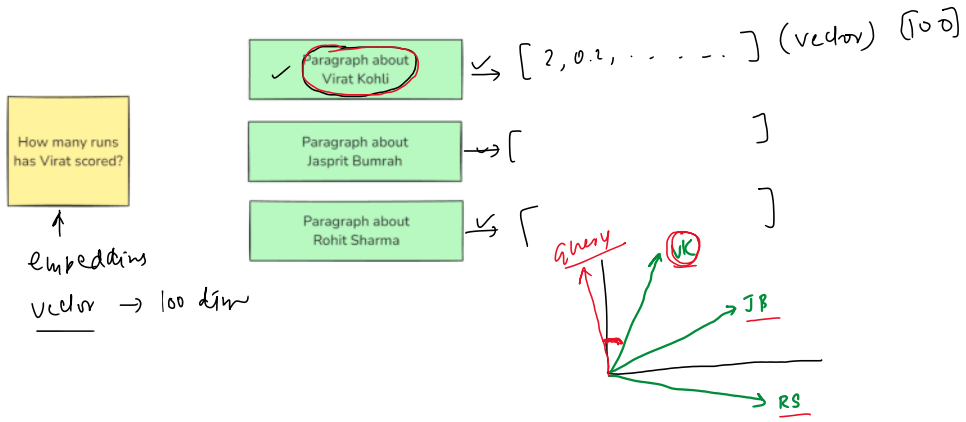
1. Explain page number 5 as if I am a 5 year old
2. Generate a True False exercise on Linear Regression
3. Generate notes for Decision Trees





Semantic search

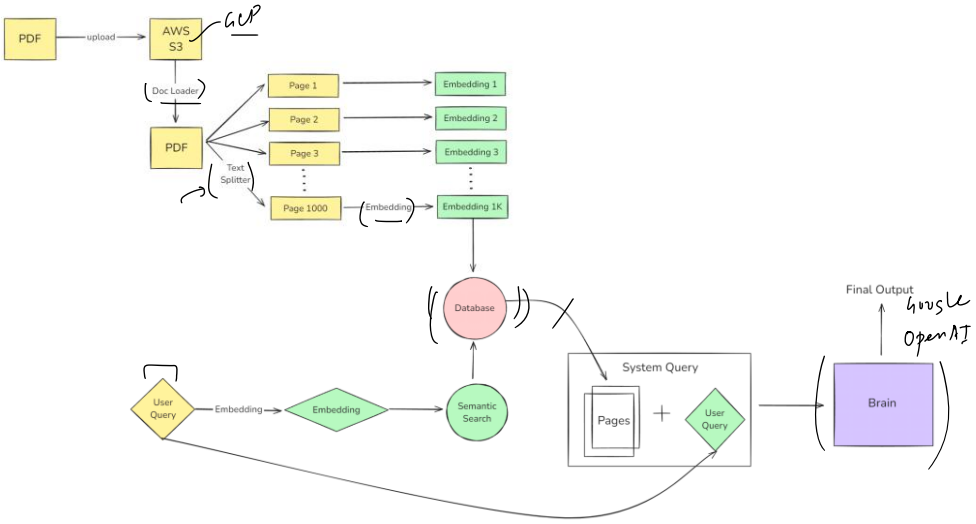
embedding → vector (set of nos.)



Benefits

21 January 2025 23:34

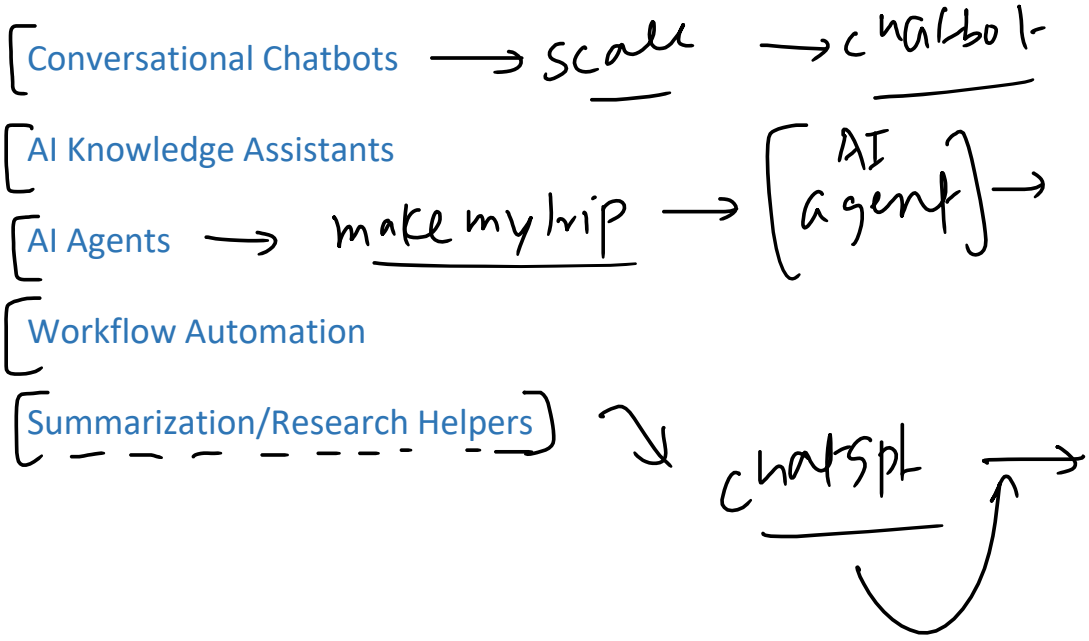
- Concept of chains
- Model Agnostic Development
- Complete ecosystem
- Memory and state handling



LLM
API

What can you build?

21 January 2025 23:34



Alternatives

21 January 2025 23:34

LlamaIndex

Haystack

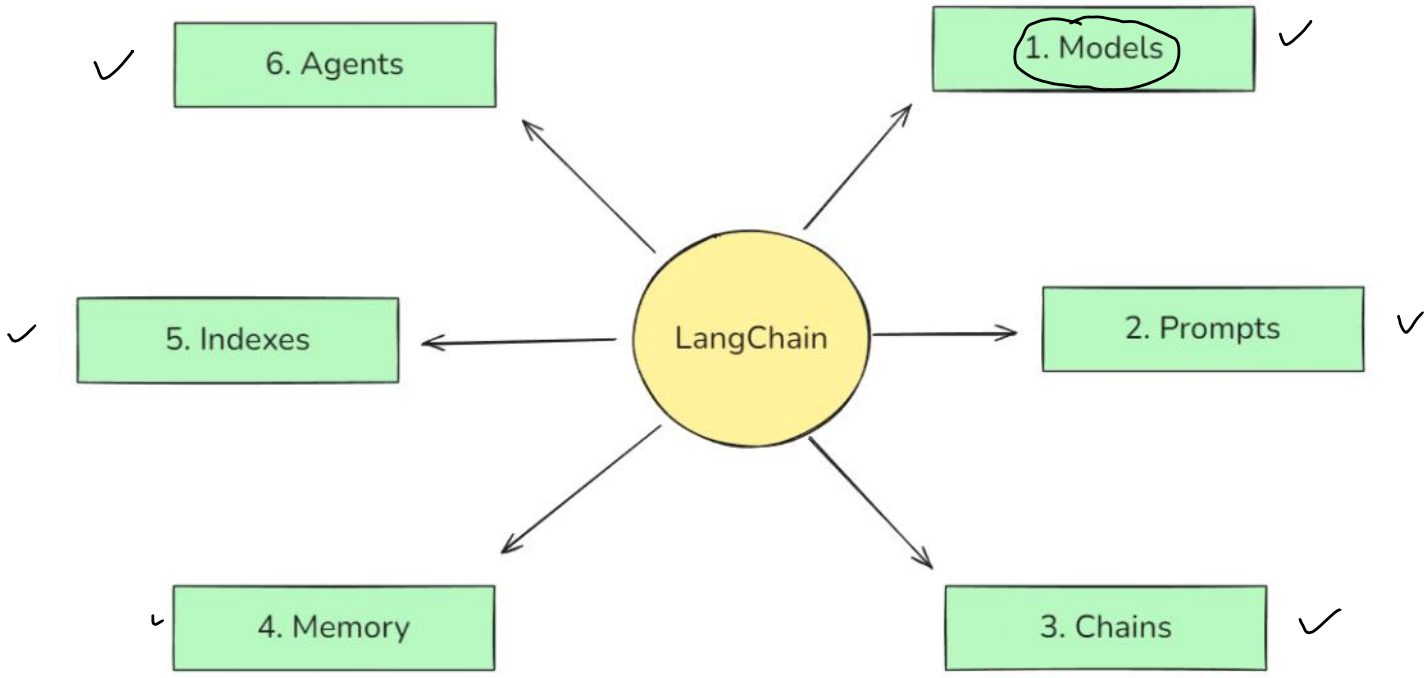
Recap

23 January 2025 10:29

LangChain is an open-source framework for developing applications powered by large language models (LLMs).

LangChain Components

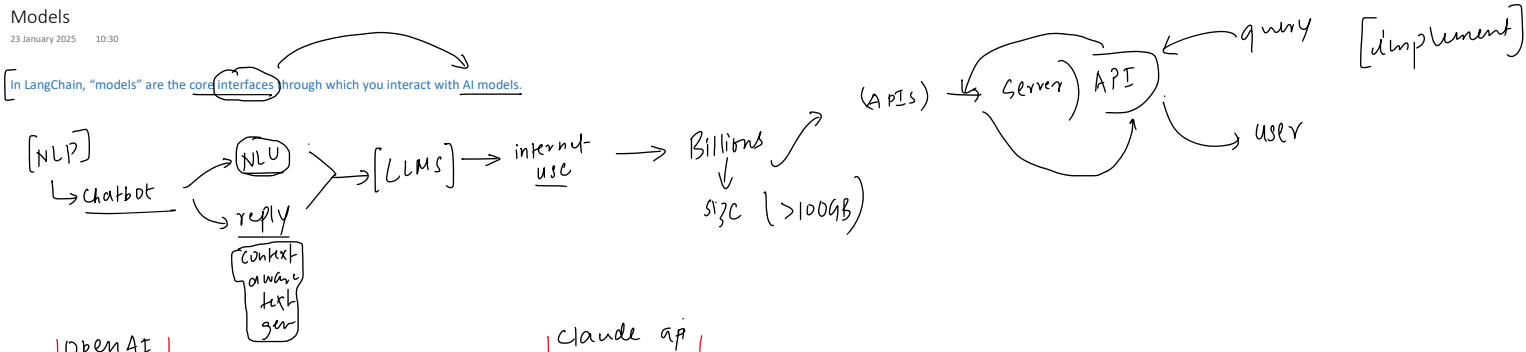
23 January 2025 10:30



Models

23 January 2025 10:30

In LangChain, "models" are the core interfaces through which you interact with AI models.



OpenAI

```python
Create a human-like response to a prompt

from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {
            "role": "user",
            "content": "Write a haiku about recursion in programming."
        }
    ]
)

print(completion.choices[0].message)
```

Claude api

```python
claude_quickstart.py

import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
    model="claude-3-5-sonnet-20241022",
    max_tokens=1000,
    temperature=0,
    system="You are a world-class poet. Respond only with short poems.",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "Why is the ocean salty?"
                }
            ]
        }
    ]
)
print(message.content)
```

Langchain

interface → API

OpenAI

```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

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

result = model.invoke("Now divide the result by 1.5")

print(result.content)
```

Langchain

Claude

```python
from langchain_anthropic import ChatAnthropic
from dotenv import load_dotenv

load_dotenv()

model = ChatAnthropic(model='claude-3-opus-20240229')

result = model.invoke("Hi who are you")

print(result.content)
```

Langchain

[language model] → LLMs

text → text

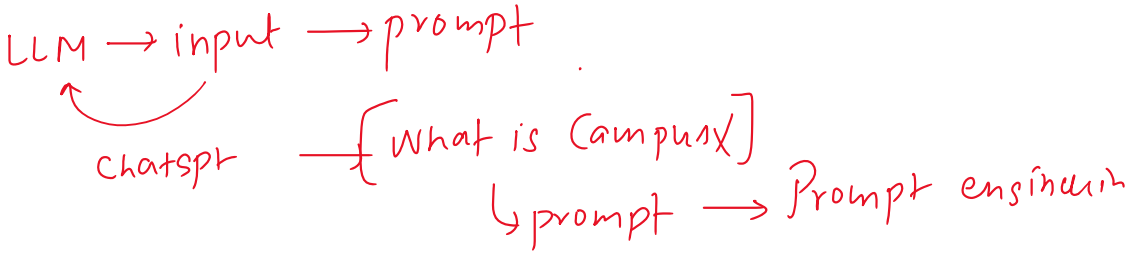
[Embedding model]

text → vector (output)

+ semantic search

Prompts

23 January 2025 10:30



1. Dynamic & Reusable Prompts

```
from langchain_core.prompts import PromptTemplate

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

print(prompt.format(topic='Cricket', length='fun'))
```

2. Role-Based Prompts

```
# Define the ChatPromptTemplate using from_template
chat_prompt = ChatPromptTemplate.from_template([
    ("system", "Hi you are a experienced {profession}"),
    ("user", "Tell me about {topic}"),
])

# Format the prompt with the variable
formatted_messages = chat_prompt.format_messages(profession="Doctor", topic="Viral Fever")
```

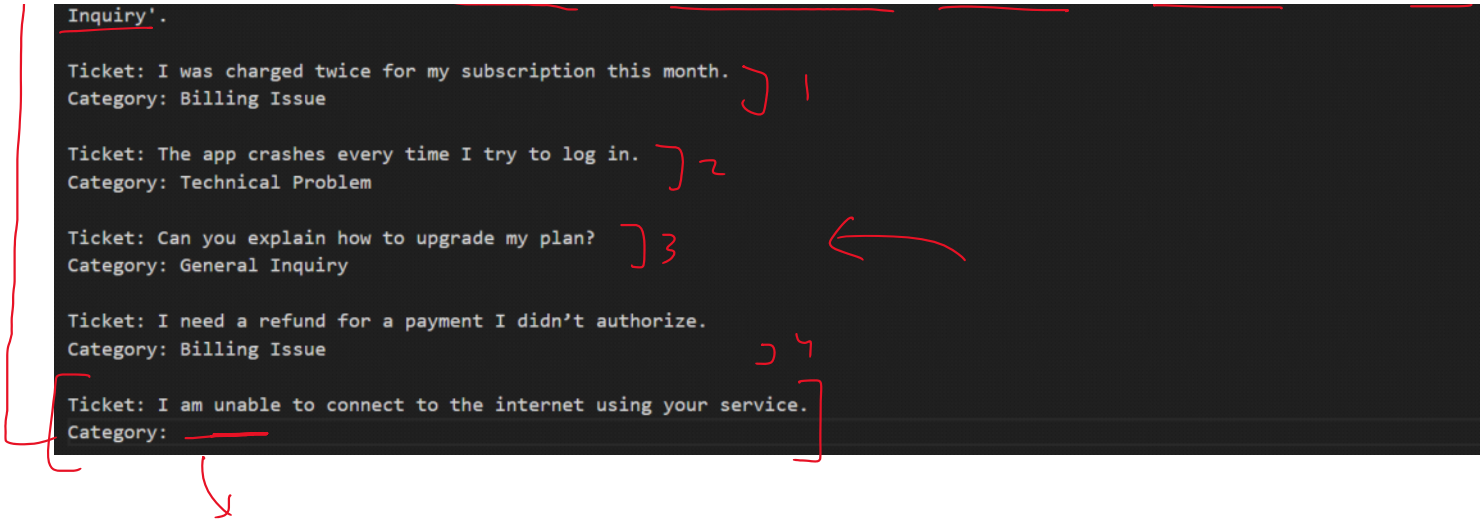
3. Few Shot Prompting

```
examples = [
    {"input": "I was charged twice for my subscription this month.", "output": "Billing Issue"},
    {"input": "The app crashes every time I try to log in.", "output": "Technical Problem"},
    {"input": "Can you explain how to upgrade my plan?", "output": "General Inquiry"},
    {"input": "I need a refund for a payment I didn't authorize.", "output": "Billing Issue"},
]
```

```
# Step 2: Create an example template
example_template = """
Ticket: {input}
Category: {output}
"""
```

```
# Step 3: 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 categories: 'Billing Issue', 'Technical Problem', or 'General Inquiry'.\n\n",
    suffix="\nTicket: {user_input}\nCategory:",
    input_variables=["user_input"],
)
```

```
Classify the following customer support tickets into one of the categories: 'Billing Issue', 'Technical Problem', or 'General Inquiry'.

Ticket: I was charged twice for my subscription this month.
Category: Billing Issue
```


Inquiry'.
Ticket: I was charged twice for my subscription this month.
Category: Billing Issue
Ticket: The app crashes every time I try to log in.
Category: Technical Problem
Ticket: Can you explain how to upgrade my plan?
Category: General Inquiry
Ticket: I need a refund for a payment I didn't authorize.
Category: Billing Issue
Ticket: I am unable to connect to the internet using your service.
Category:

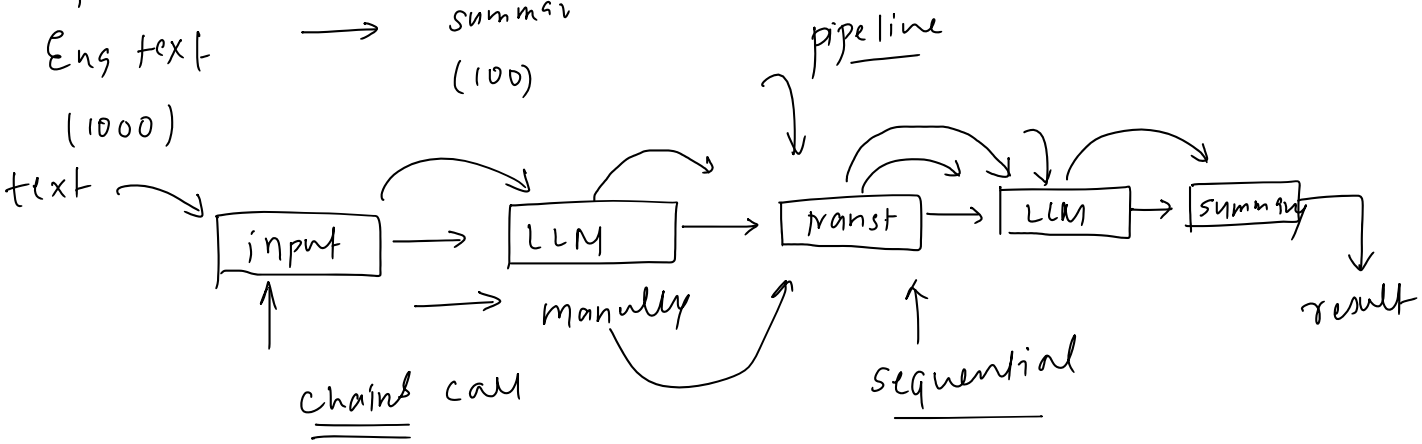
Chains →

23 January 2025 10:31

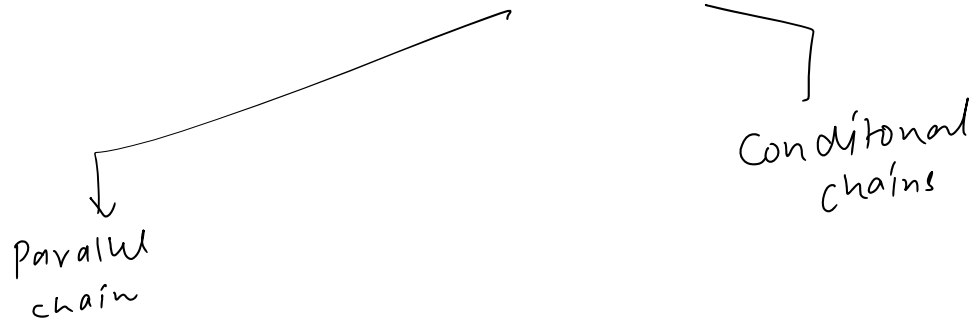
Pipelines → LLM + Pipeline

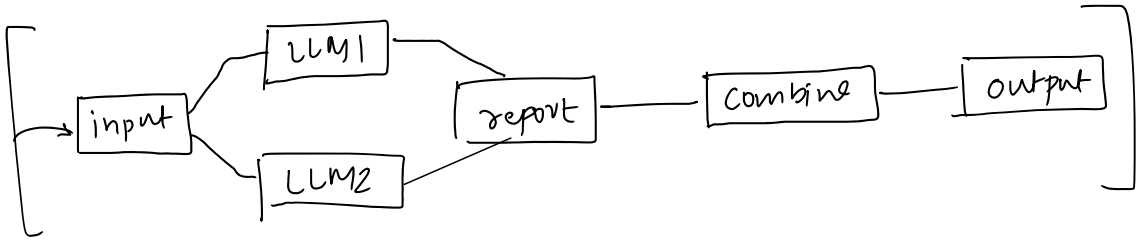
input

Eng text (1000) → Hindi summary (100)



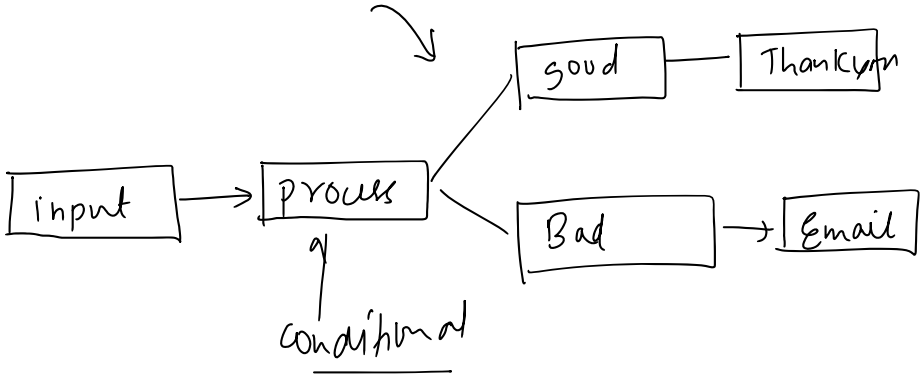
Complex pipeline





AI Agent

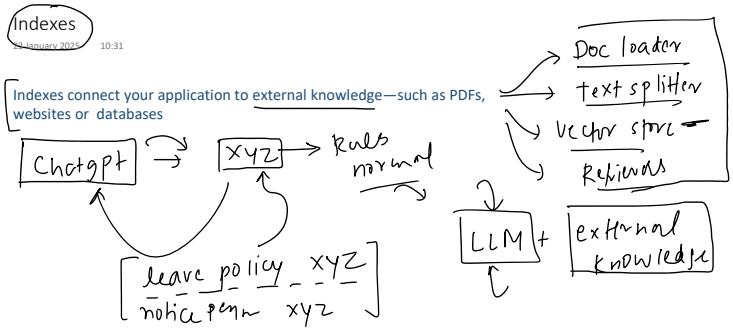
feedback



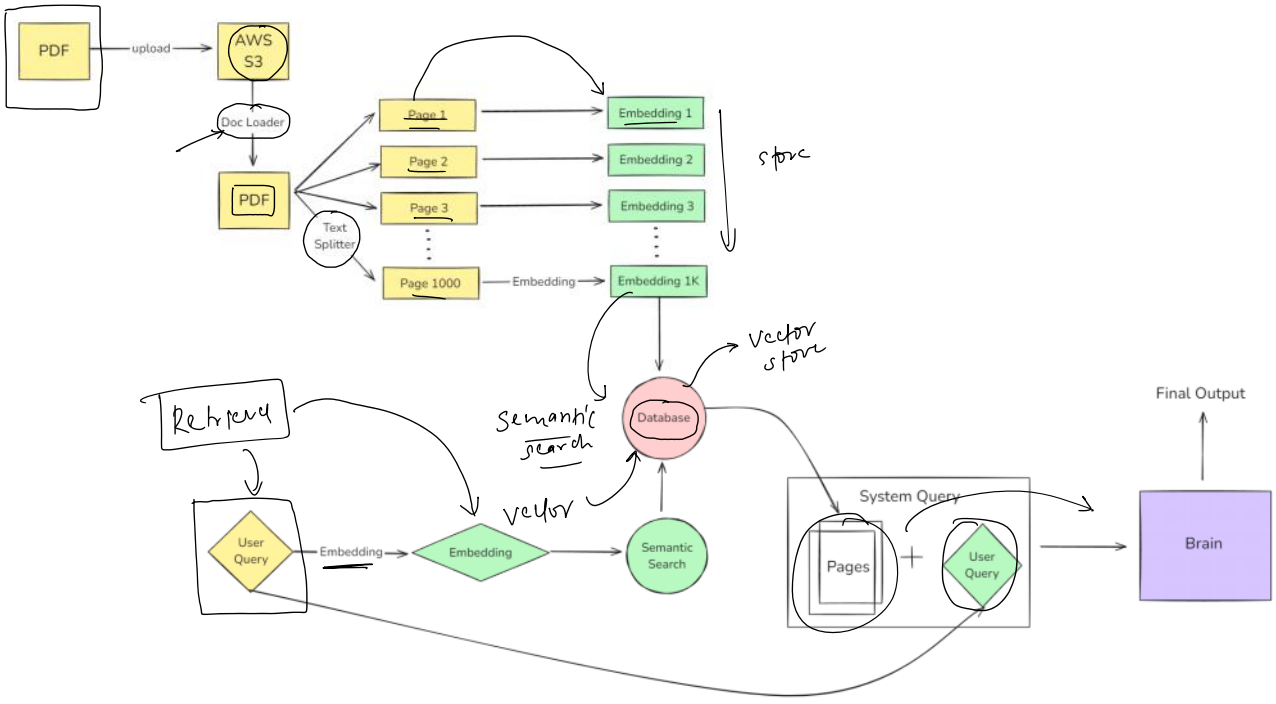
Indexes

23 January 2025 10:31

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



Rulebook 1000 pages



LLM
API

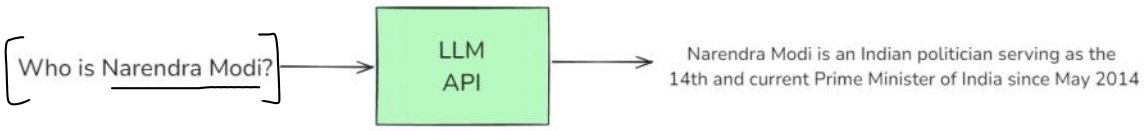
Memory

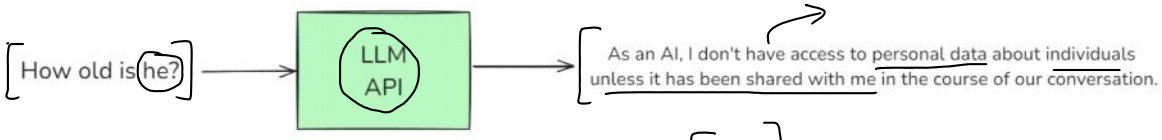
23 January 2025 10:31

LLM API calls are stateless

Chatbot

GPT

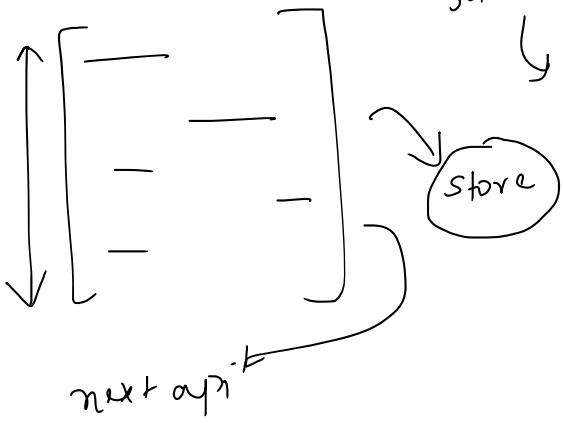




last n intrs [100]

summary → api call

- ConversationBufferMemory: Stores a transcript of recent messages. Great for short chats but can grow large quickly.
- ConversationBufferWindowMemory: Only keeps the last N interactions to avoid excessive token usage.
- Summarizer-Based Memory: Periodically summarizes older chat segments to keep a condensed memory footprint.
- Custom Memory: For advanced use cases, you can store specialized state (e.g., the user's preferences or key facts about them) in a custom memory class.



Agents

23 January 2025 10:31

AI Agents

- LLMs (NLU + Textgen) → chatbot

Chatbot → AI agent → Chatbot with super

AI agent

travel website

- Shimla
- ~~Manali~~

→ 24th Jan

- API
  - Indigo
  - Book the flight

AI agent-

- [Reasoning capability]
- TOOLS

Tools → Calculator, Weather API

User → AI agent → Can you multiply today's temp of Delhi with 3

- Chain of thought

→ [Delhi temp]

→ 25°C

→ 25, 3, *

75 → (75)

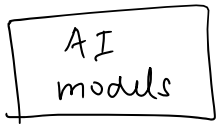
Recap

10 February 2025 09:21

+ Models

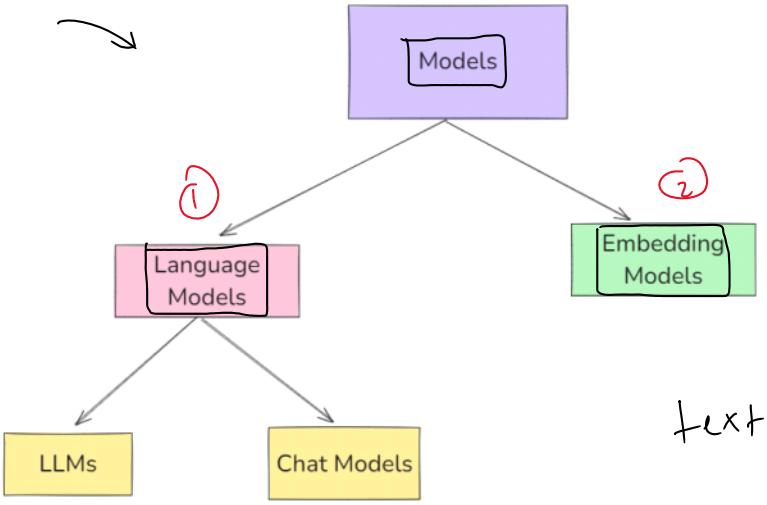
What are Models

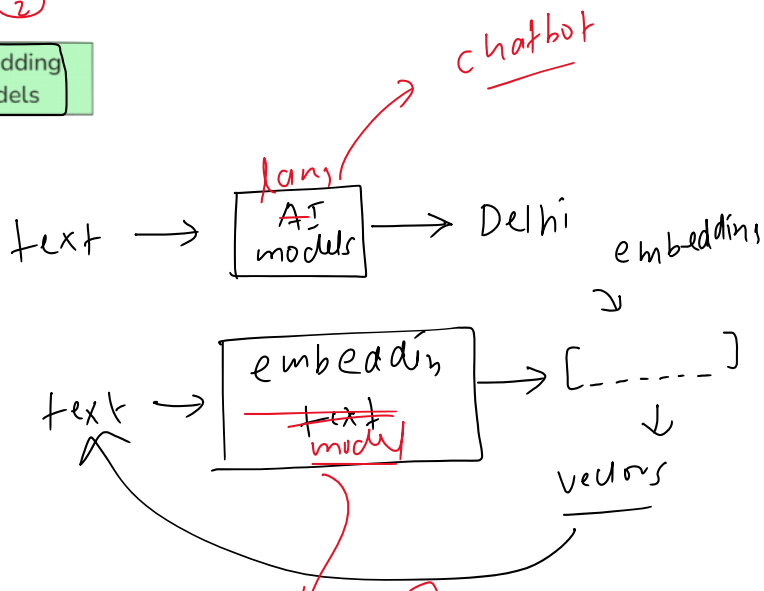
07 January 2025 23:15

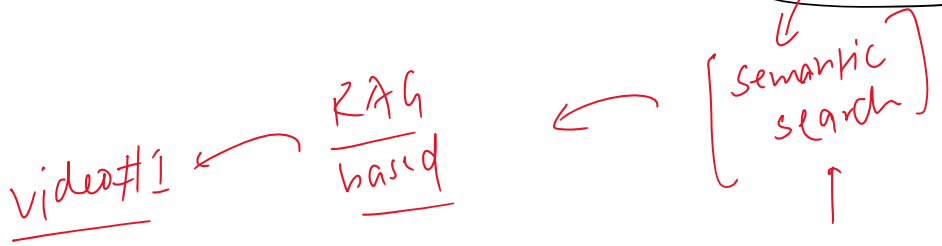


The Model Component in LangChain is a crucial part of the framework, designed to facilitate interactions with various language models and embedding models.

It abstracts the complexity of working directly with different LLMs, chat models, and embedding models, providing a uniform interface to communicate with them. This makes it easier to build applications that rely on AI-generated text, text embeddings for similarity search, and retrieval-augmented generation (RAG).

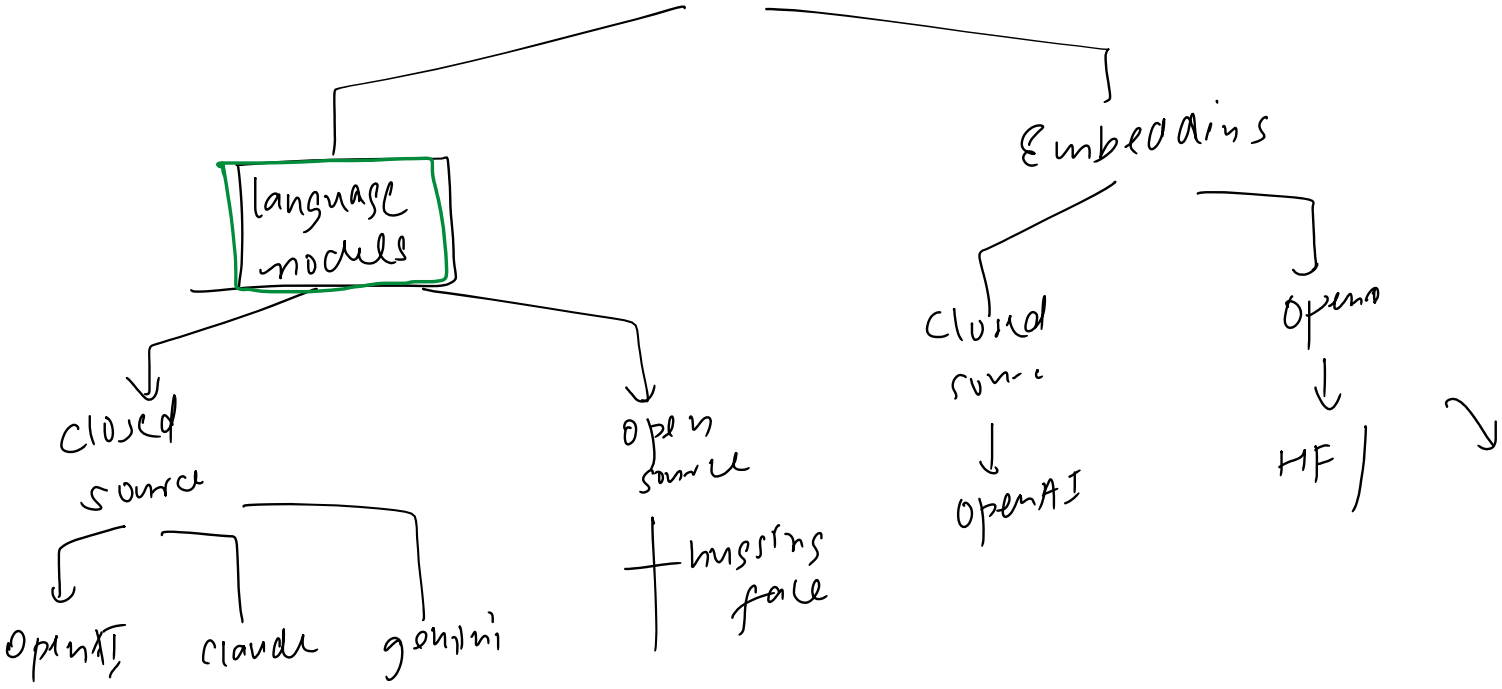






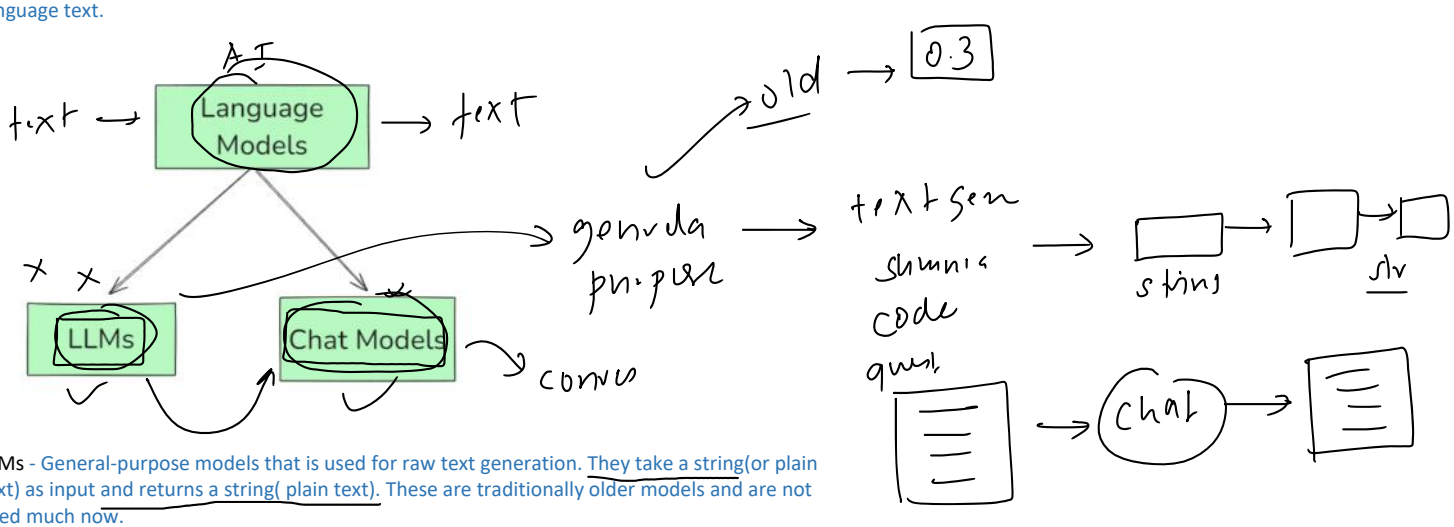
Plan of Action

10 February 2025 09:21



Language Models

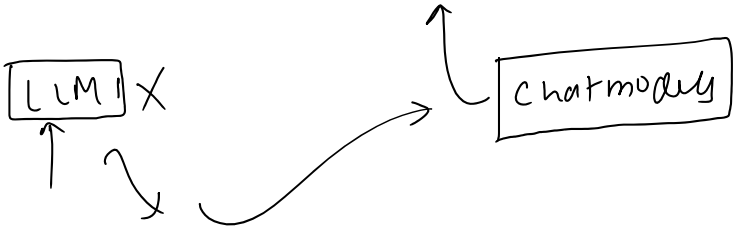
Language Models are AI systems designed to process, generate, and understand natural language text.



LLMs - General-purpose models that is used for raw text generation. They take a string(or plain text) as input and returns a string(plain text). These are traditionally older models and are not used much now.

Chat Models - Language models that are specialized for conversational tasks. They take a sequence of messages as inputs and return chat messages as outputs (as opposed to using plain text). These are traditionally newer models and used more in comparison to the LLMs.

Feature	LLMs (Base Models)	Chat Models (Instruction-Tuned)
Purpose	Free-form text generation	Optimized for multi-turn conversations
Training Data	General text corpora (books, articles)	Fine-tuned on chat datasets (dialogues, user-assistant conversations)
Memory & Context	No built-in memory	Supports structured conversation history
Role Awareness	No understanding of "user" and "assistant" roles	Understands "system", "user", and "assistant" roles
Example Models	GPT-3, Llama-2-7B, Mistral-7B, OPT-1.3B	GPT-4, GPT-3.5-turbo, Llama-2-Chat, Mistral-Instruct, Claude
Use Cases	Text generation, summarization, translation, creative writing, code generation	Conversational AI, chatbots, virtual assistants, customer support, AI tutors

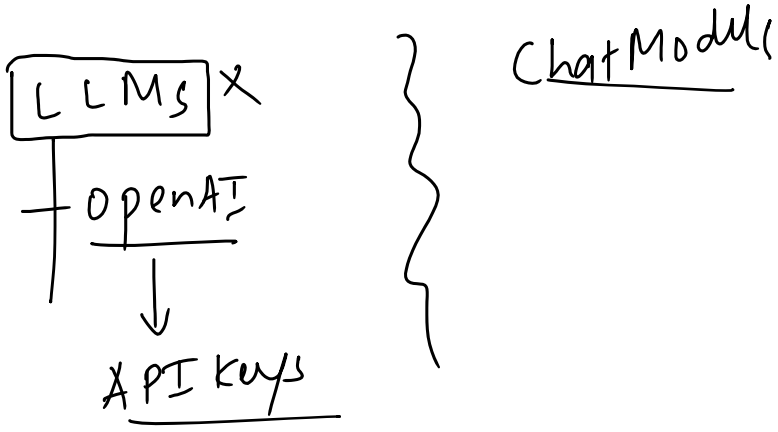


Setup

10 February 2025 09:21

Demo

1. OpenAI
2. Anthropic
3. Google
4. HuggingFace



`temperature` is a parameter that **controls the randomness of a language model's output**. It affects how **creative or deterministic** the responses are.

- **Lower values** (`0.0 - 0.3`) → More **deterministic** and predictable.
- **Higher values** (`0.7 - 1.5`) → More **random**, creative, and diverse.

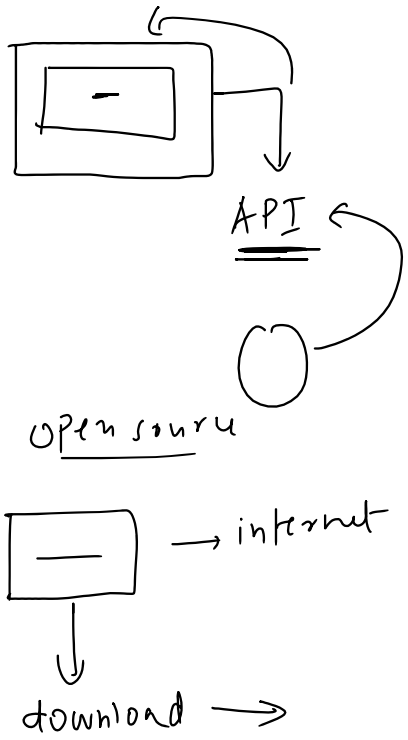
Use Case	Recommended Temperature
Factual answers (math, code, facts)	`0.0 - 0.3`
Balanced response (general QA, explanations)	`0.5 - 0.7`
Creative writing, storytelling, jokes	`0.9 - 1.2`
Maximum randomness (wild ideas, brainstorming)	`1.5+`

Open Source Models

11 February 2025 08:58

Open-source language models are freely available AI models that can be downloaded, modified, fine-tuned, and deployed without restrictions from a central provider. Unlike closed-source models such as OpenAI's GPT-4, Anthropic's Claude, or Google's Gemini, open-source models allow full control and customization.

Feature	Open-Source Models	Closed-Source Models
Cost	Free to use (no API costs)	Paid API usage (e.g., OpenAI charges per token)
Control	Can modify, fine-tune, and deploy anywhere	Locked to provider's infrastructure
Data Privacy	Runs locally (no data sent to external servers)	Sends queries to provider's servers
Customization	Can fine-tune on specific datasets	No access to fine-tuning in most cases
Deployment	Can be deployed on **on-premise** servers or **cloud**	Must use vendor's API



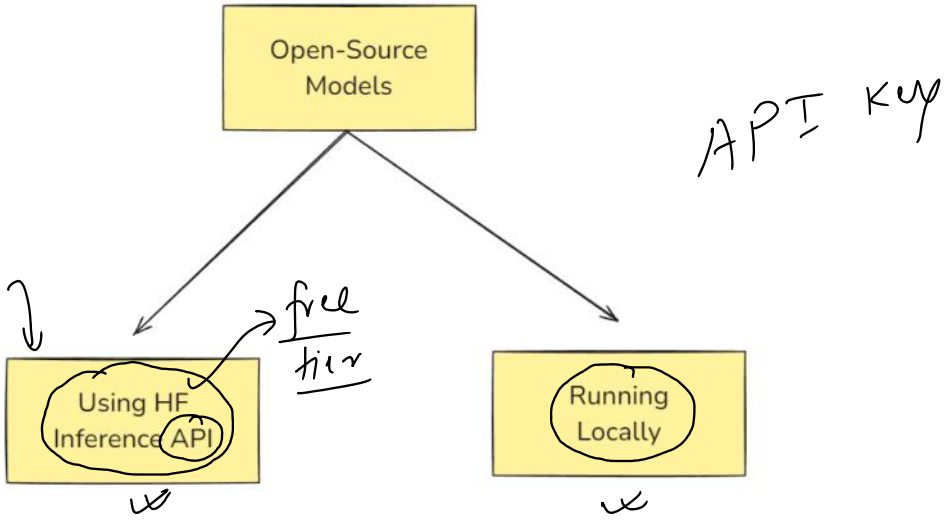
Some Famous Open Source Models

Model	Developer	Parameters	Best Use Case
LLaMA-2-7B/13B/70B	Meta AI	7B - 70B	General-purpose text generation
Mixtral-8x7B	Mistral AI	8x7B (MoE)	Efficient & fast responses
Mistral-7B	Mistral AI	7B	Best small-scale model (outperforms LLaMA-2-13B)
Falcon-7B/40B	TII UAE	7B - 40B	High-speed inference
BLOOM-176B	BigScience	176B	Multilingual text generation
GPT-J-6B	EleutherAI	6B	Lightweight and efficient
GPT-NeoX-20B	EleutherAI	20B	Large-scale applications
StableLM	Stability AI	3B - 7B	Compact models for chatbots

Where to find them?

HuggingFace - The largest repository of open-source LLMs

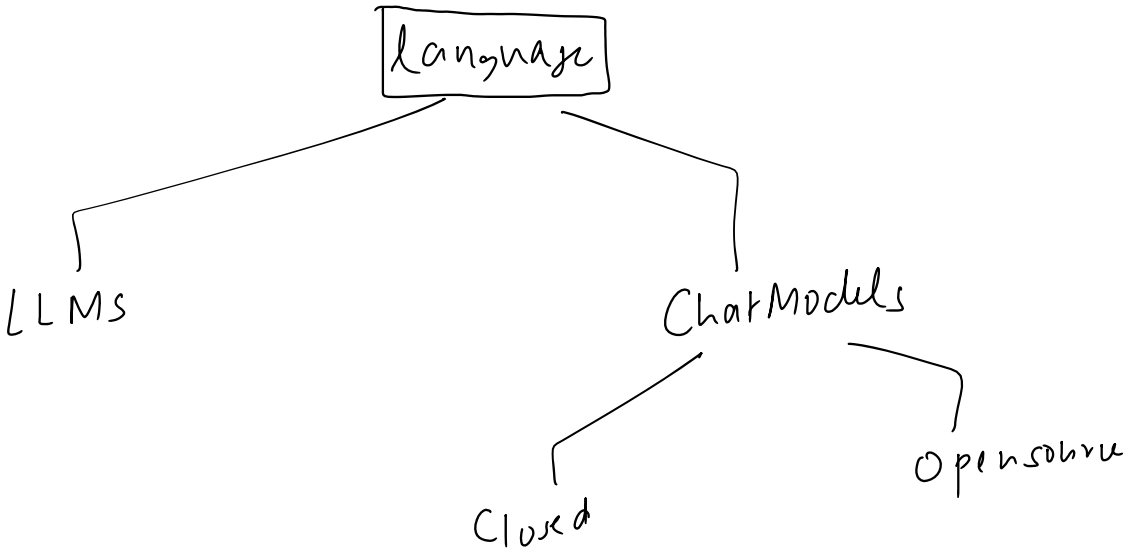
Ways to use Open-source Models

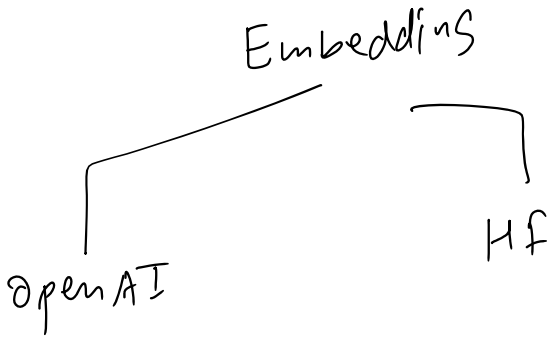


Disadvantages

Disadvantage	Details
High Hardware	Running large models (e.g., LLaMA-2-70B) requires expensive GPUs.

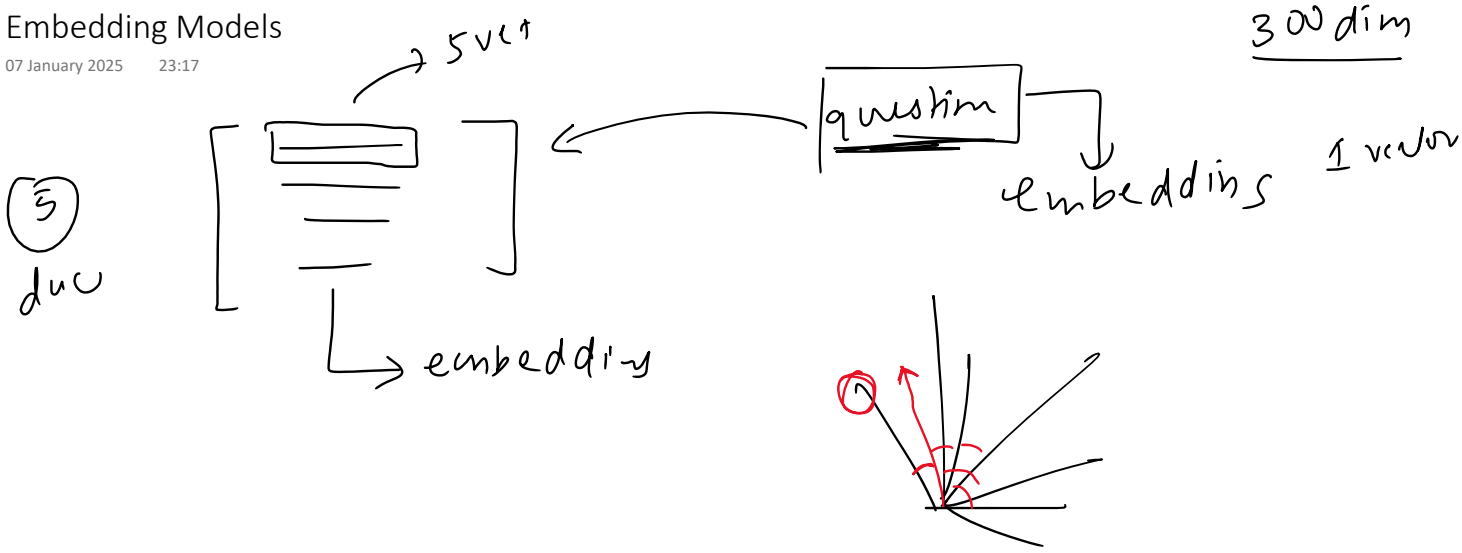
Disadvantage	Details
High Hardware Requirements	Running large models (e.g., LLaMA-2-70B) requires expensive GPUs.
Setup Complexity	Requires installation of dependencies like **PyTorch**, **CUDA**, **transformers**.
Lack of RLHF	Most open-source models don't have **fine-tuning with human feedback**, making them weaker in instruction-following.
Limited Multimodal Abilities	Open models don't support **images, audio, or video** like GPT-4V.





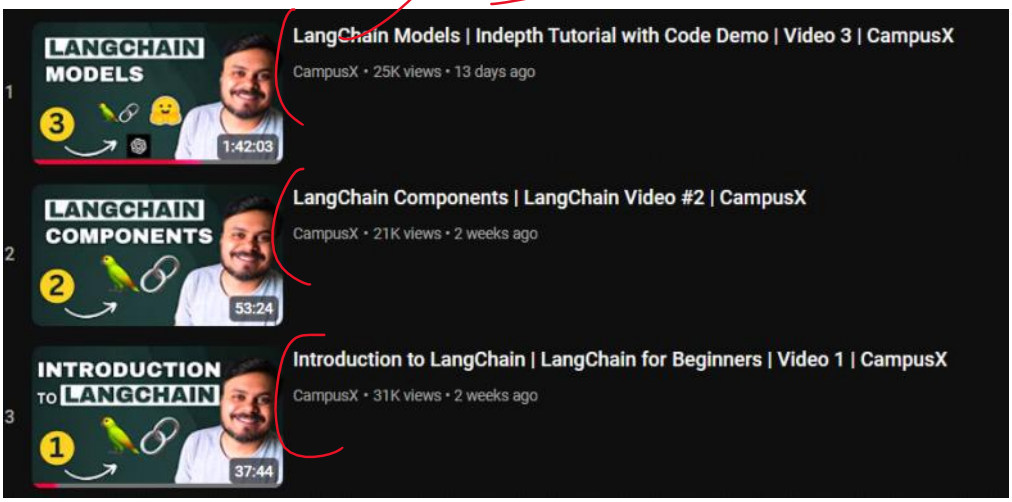
Embedding Models

07 January 2025 23:17



Recap

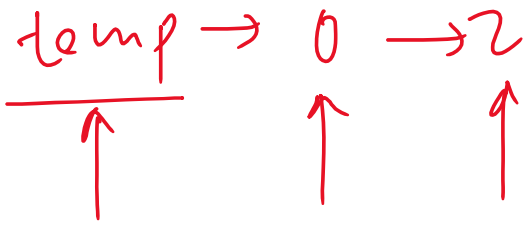
24 February 2025 09:42



④ → Prompts

A mistake from my side!

24 February 2025 09:42

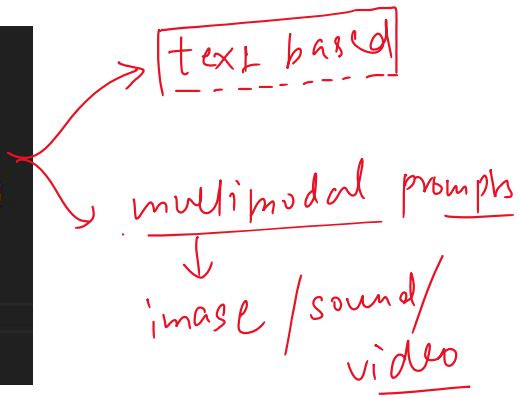


What are Prompts

10 January 2025 08:37

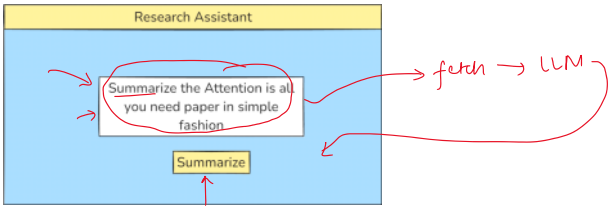
Prompts are the input instructions or queries given to a model to guide its output.

```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

model = ChatOpenAI(model='gpt-4', temperature=1.5, max_completion_tokens=10)

result = model.invoke("Write a 5 line poem on cricket")

print(result.content)
```



Static vs Dynamic Prompts

14 February 2025 00:01




```
paper_input = st.selectbox( "Select Research Paper Name", ["Select...", "Attention Is All You Need",
"BERT: Pre-training of Deep Bidirectional Transformers", "GPT-3: Language Models are Few-Shot
Learners", "Diffusion Models Beat GANs on Image Synthesis"] )

style_input = st.selectbox( "Select Explanation Style", ["Beginner-Friendly", "Technical", "Code-
Oriented", "Mathematical"] )

length_input = st.selectbox( "Select Explanation Length", ["Short (1-2 paragraphs)", "Medium (3-5
paragraphs)", "Long (detailed explanation)"] )
```

A **PromptTemplate** in LangChain is a structured way to create prompts dynamically by inserting variables into a predefined template. Instead of hardcoding prompts, PromptTemplate allows you to define placeholders that can be filled in at runtime with different inputs.

This makes it reusable, flexible, and easy to manage, especially when working with dynamic user inputs or automated workflows.

Prompt Template

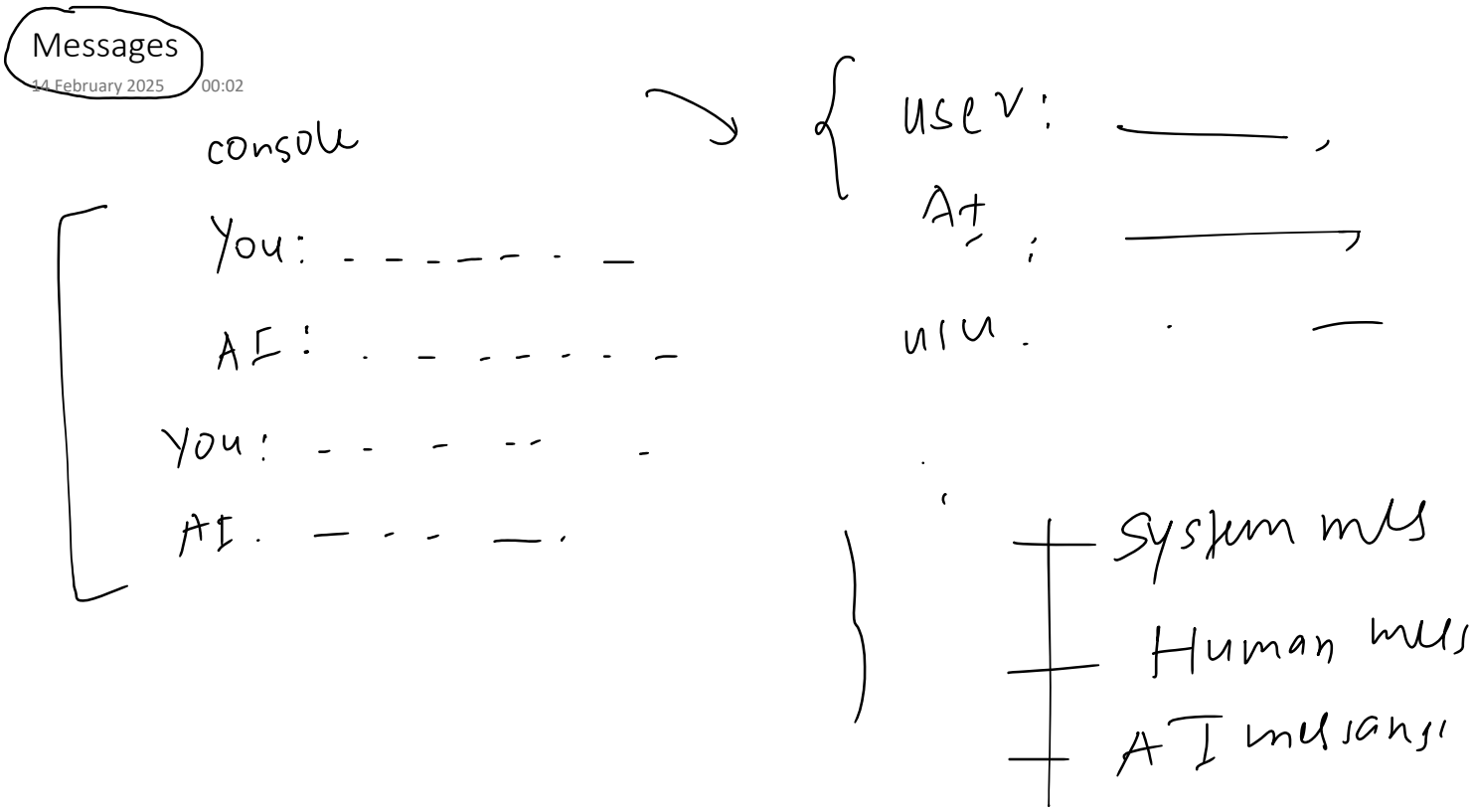
10 January 2025 08:37

A **PromptTemplate** in LangChain is a structured way to create prompts dynamically by inserting variables into a predefined template. Instead of hardcoding prompts, PromptTemplate allows you to define placeholders that can be filled in at runtime with different inputs.

This makes it reusable, flexible, and easy to manage, especially when working with dynamic user inputs or automated workflows.

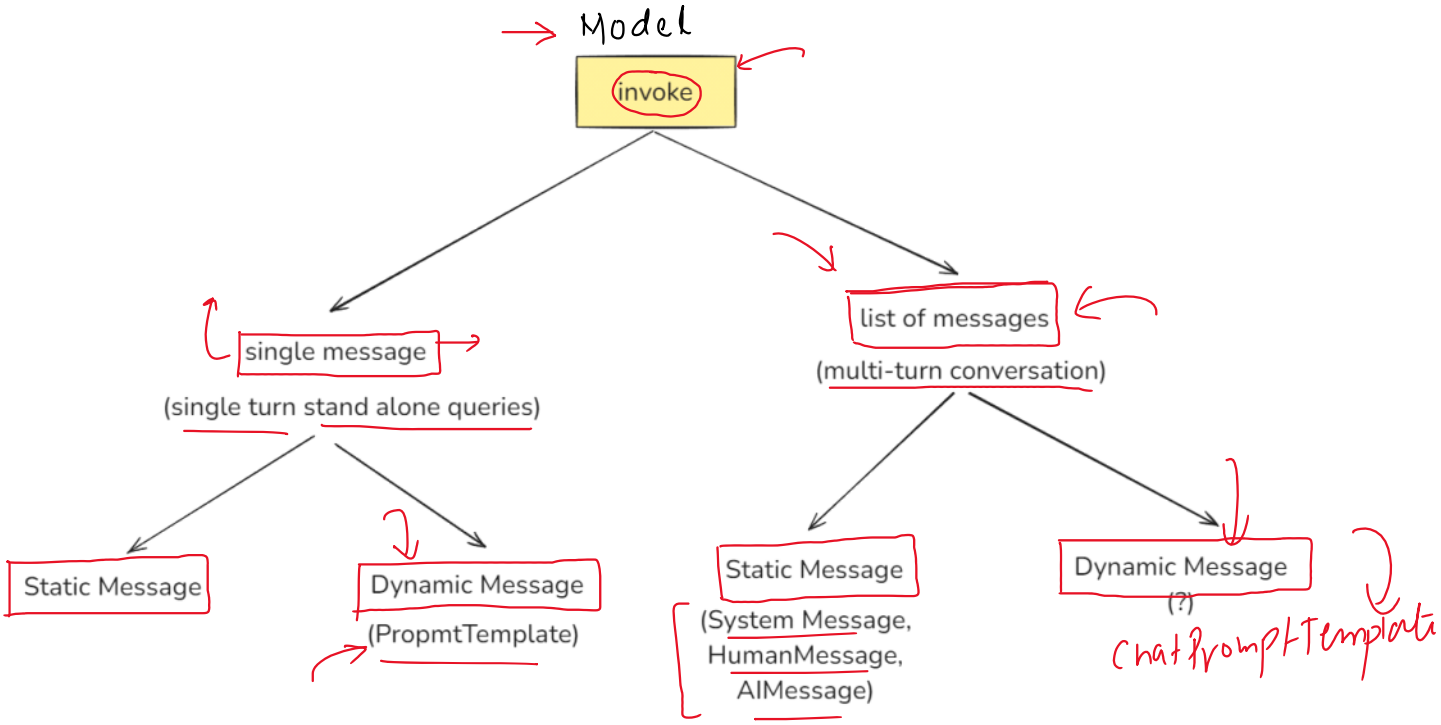
Why use PromptTemplate over f strings?

1. Default validation
2. Reusable
3. LangChain Ecosystem


Messages
14 February 2025 00:02
console
You:
AI:
You:
AI.
{ user:
At
system msg
Human msg
AI msg

Chat Prompt Templates

14 February 2025 00:02



System Message → dynamic

[You are a helpful {domain} expert

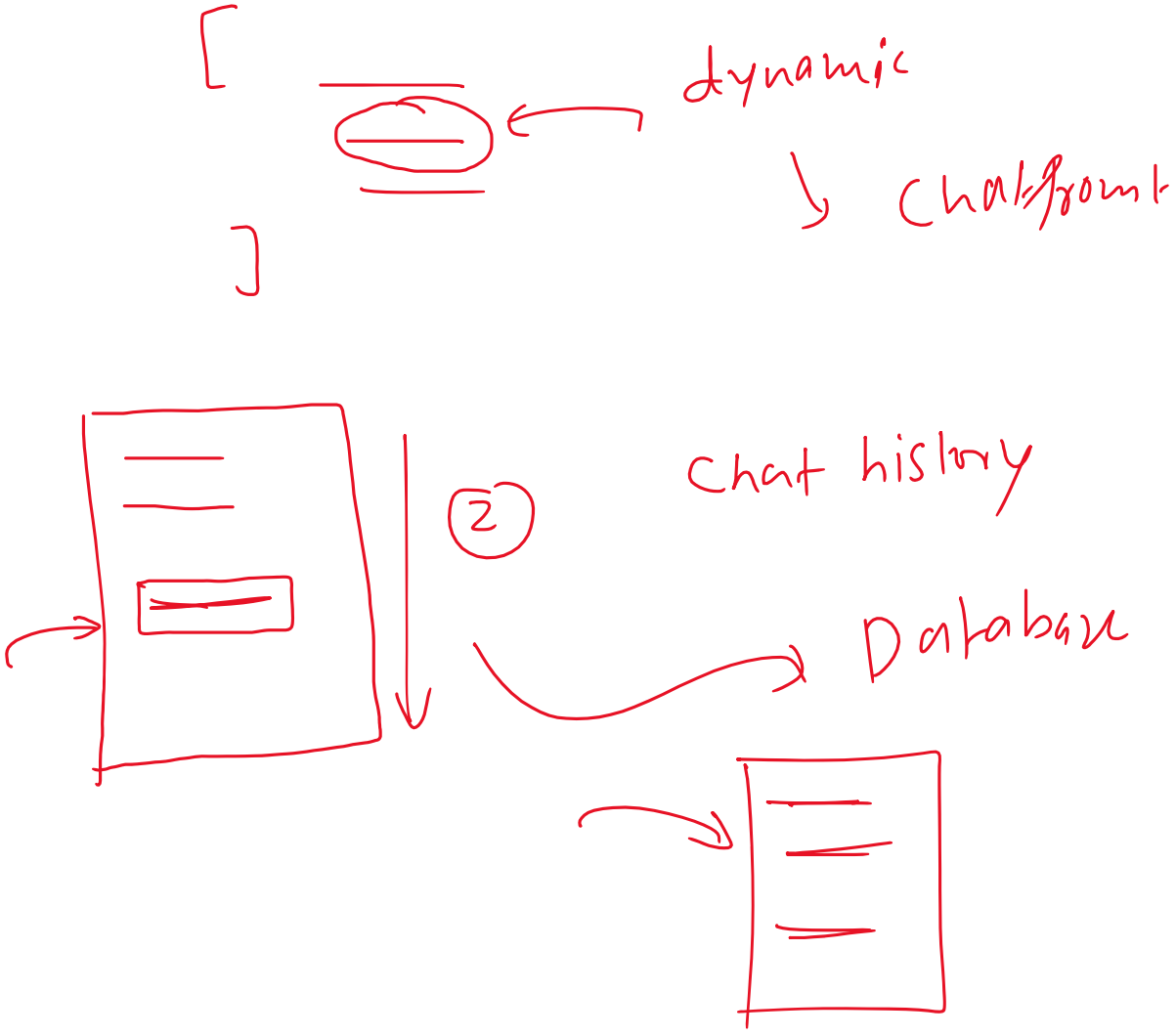
Human me

Explain about {topic}

Message Placeholder

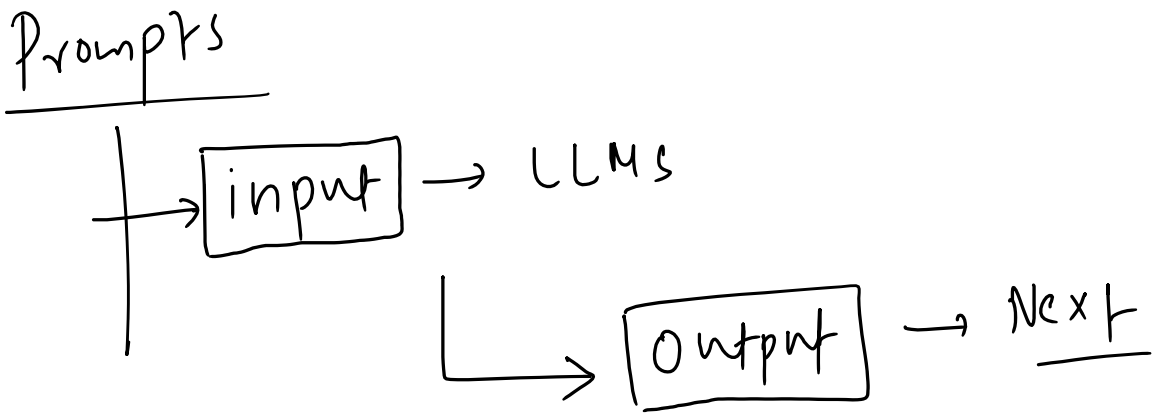
14 February 2025 00:09

A MessagesPlaceholder in LangChain is a special placeholder used inside a ChatPromptTemplate to dynamically insert chat history or a list of messages at runtime.



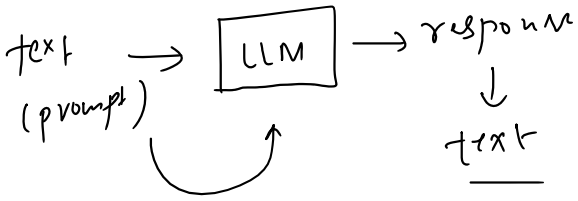
Recap

28 February 2025 00:12



Structured Output

28 February 2025 00:13



In LangChain, structured output refers to the practice of having language models return responses in a well-defined data format (for example, JSON), rather than free-form text. This makes the model output easier to parse and work with programmatically.

[Prompt] - *Can you create a one-day travel itinerary for Paris?*

[LLM's Unstructured Response]

Here's a suggested itinerary: *Morning: Visit the Eiffel Tower.*

Afternoon: Walk through the Louvre Museum.

Evening: Enjoy dinner at a Seine riverside café.

[JSON enforced output]

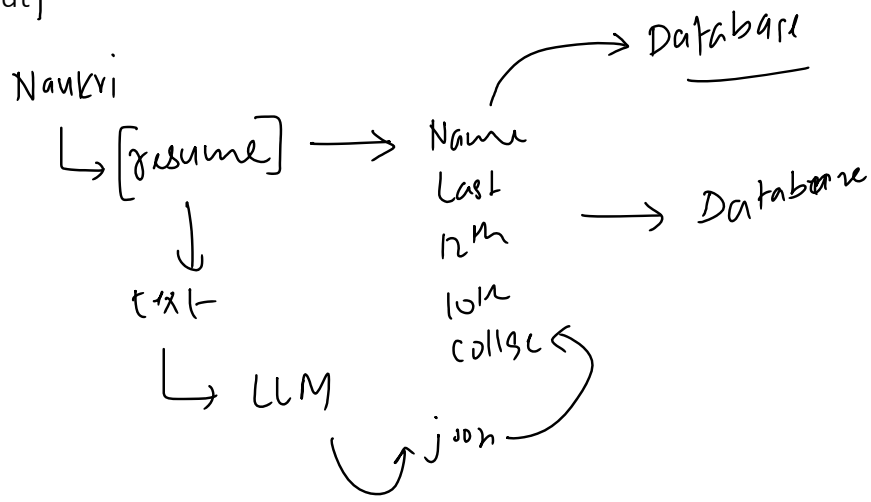
```
[
  {"time": "Morning", "activity": "Visit the Eiffel Tower"},
  {"time": "Afternoon", "activity": "Walk through the Louvre Museum"},
  {"time": "Evening", "activity": "Enjoy dinner at a Seine riverside café"}
]
```

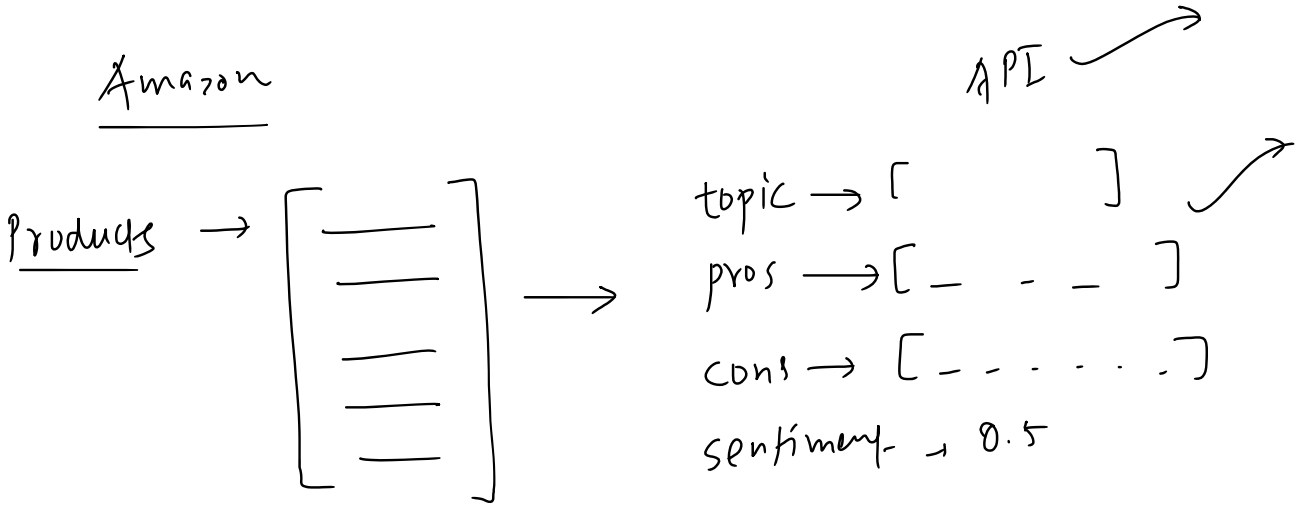
Source - https://www.linkedin.com/pulse/structured-outputs-from-llms-langchain-output-parsers-vijay-chaudhary-wgjqc

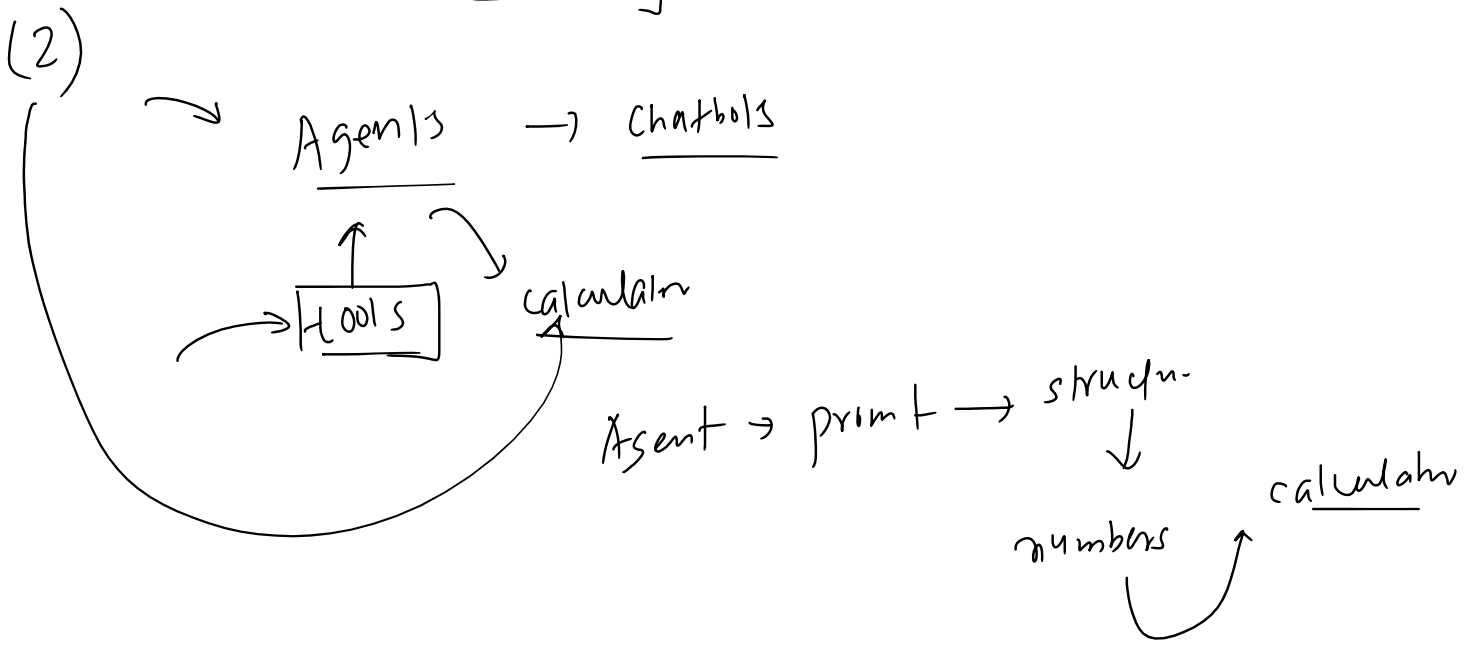
Why do we need Structured Output

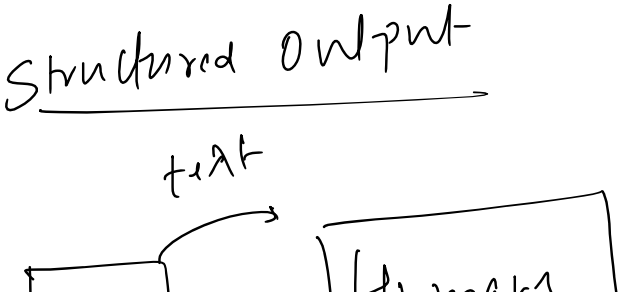
28 February 2025 00:13

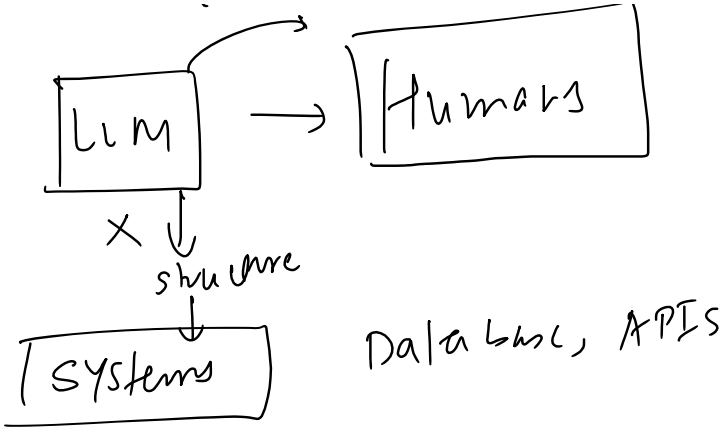
Data Extraction
API building
Agents





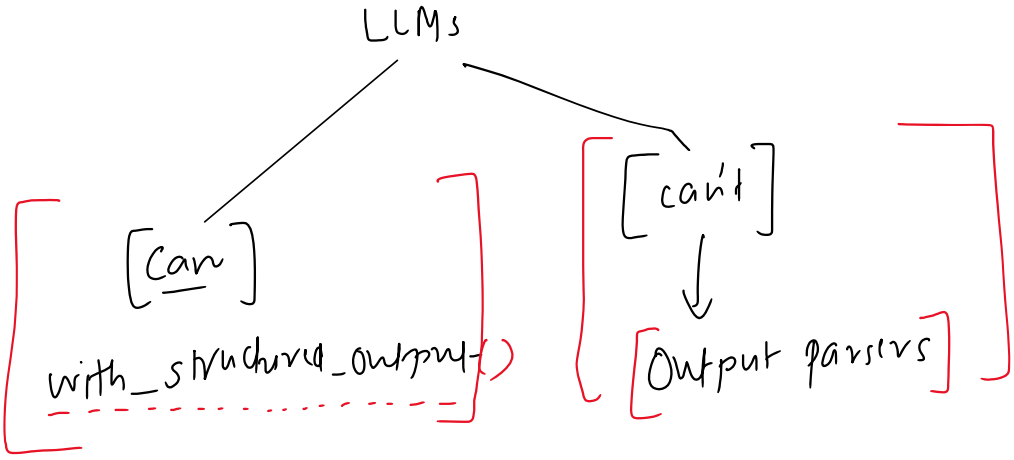





LLM
Humans
structure
Systems
Databases, APIs

Ways to get Structured Output

28 February 2025 00:14

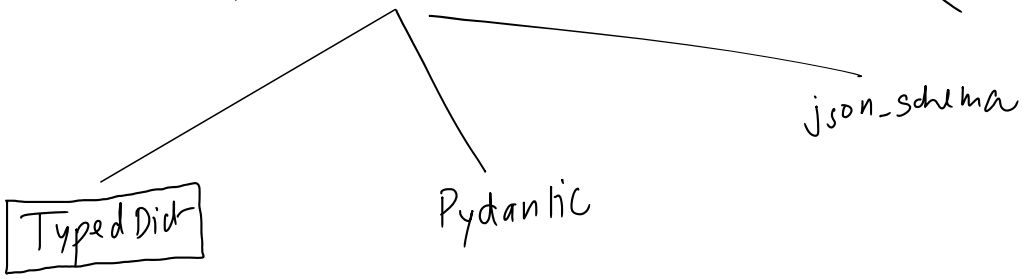


with_structured_output) → data-format

28 February 2025 00:15

(model_invoke)

You are an AI assistant that extracts structured insights from text. Given a product review, extract: - Summary: A brief overview of the main points. - Sentiment: Overall tone of the review (positive, neutral, negative). Return the response in JSON format.



TypedDict

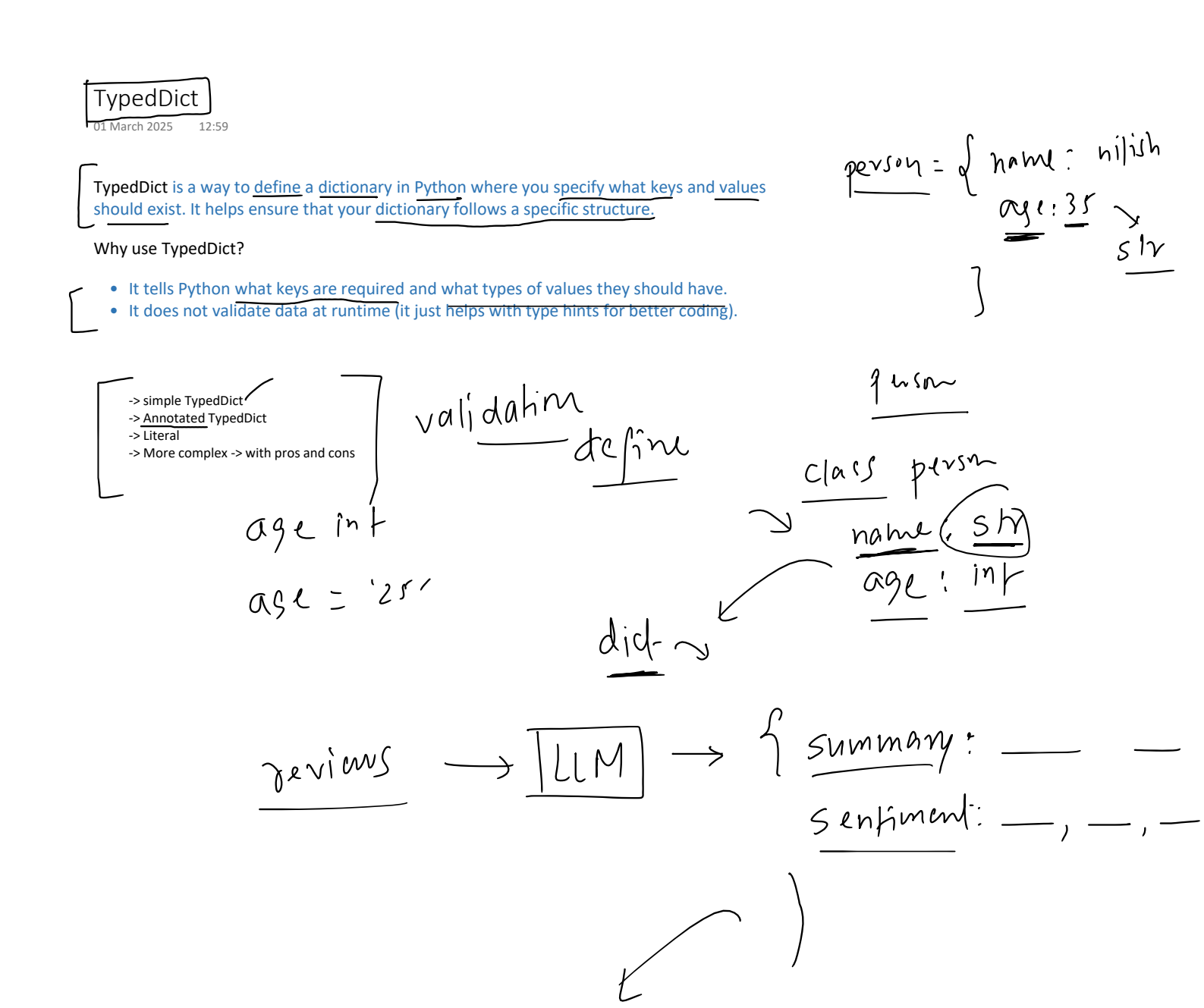
01 March 2025 12:59

TypedDict is a way to define a dictionary in Python where you specify what keys and values should exist. It helps ensure that your dictionary follows a specific structure.

Why use TypedDict?

- It tells Python what keys are required and what types of values they should have.
- It does not validate data at runtime (it just helps with type hints for better coding).

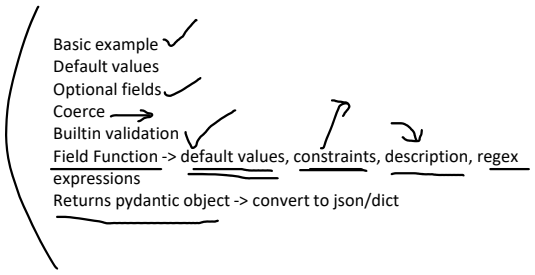
-> simple TypedDict
-> Annotated TypedDict
-> Literal
-> More complex -> with pros and cons



Pydantic

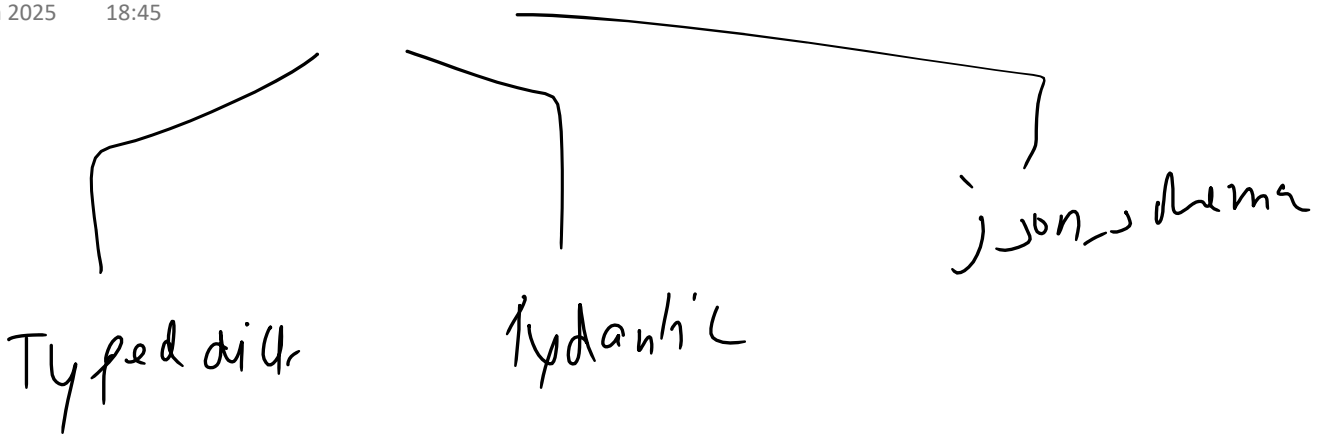
01 March 2025 12:59

Pydantic is a data validation and data parsing library for Python. It ensures that the data you work with is correct, structured, and type-safe.



Json

01 March 2025 18:45



When to use what?

01 March 2025 12:59

✅ Use `TypedDict` if:

- You **only need type hints** (basic structure enforcement).
- You **don't need validation** (e.g., checking numbers are positive).
- You **trust the LLM** to return correct data.

✅ Use `Pydantic` if:

- You need **data validation** (e.g., sentiment must be `"positive"`, `"neutral"`, or `"negative"`).
- You need **default values** if the LLM **misses fields**.
- You want **automatic type conversion** (e.g., `"100"` → `100`).

✅ Use `JSON Schema` if:

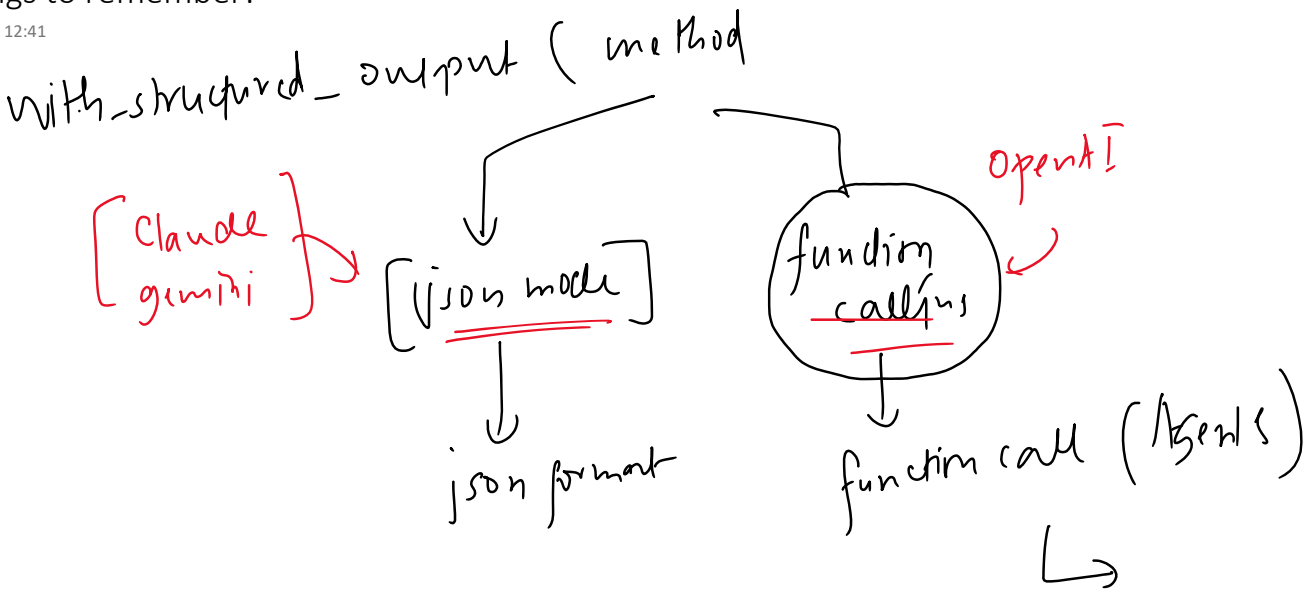
- You **don't want to ~~import extra Python libraries~~** (Pydantic).
- You need **validation but don't need Python objects**.
- You want to define structure **in a standard JSON format**.

🚀 When to Use What?

Feature	TypedDict ✅	Pydantic 🚀	JSON Schema 🌍
Basic structure	✅	✅	✅
Type enforcement	✅	✅	✅
Data validation	❌	✅	✅
Default values	❌	✅	❌
Automatic conversion	❌	✅	❌
Cross-language compatibility	❌	❌	✅

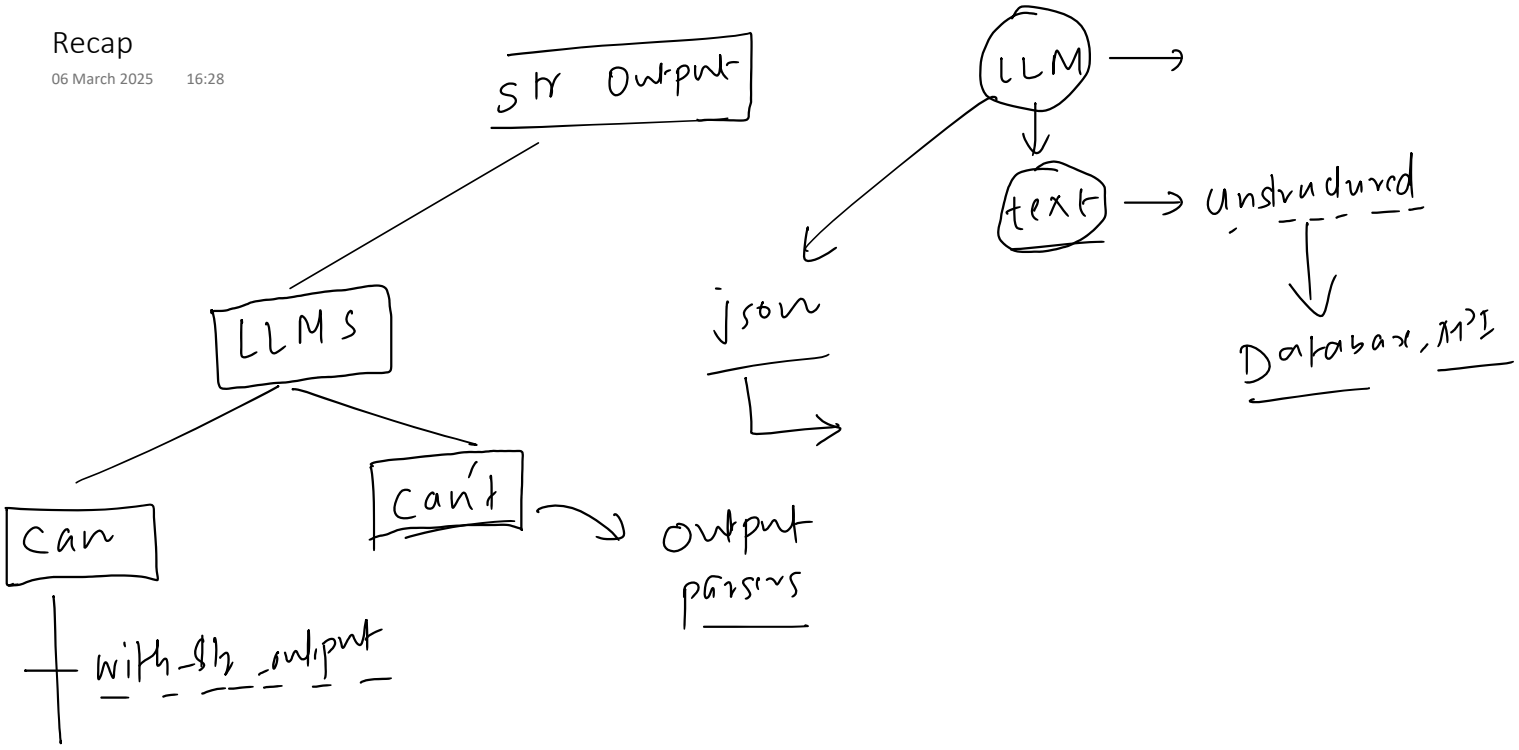
A few things to remember!

03 March 2025 12:41



Recap

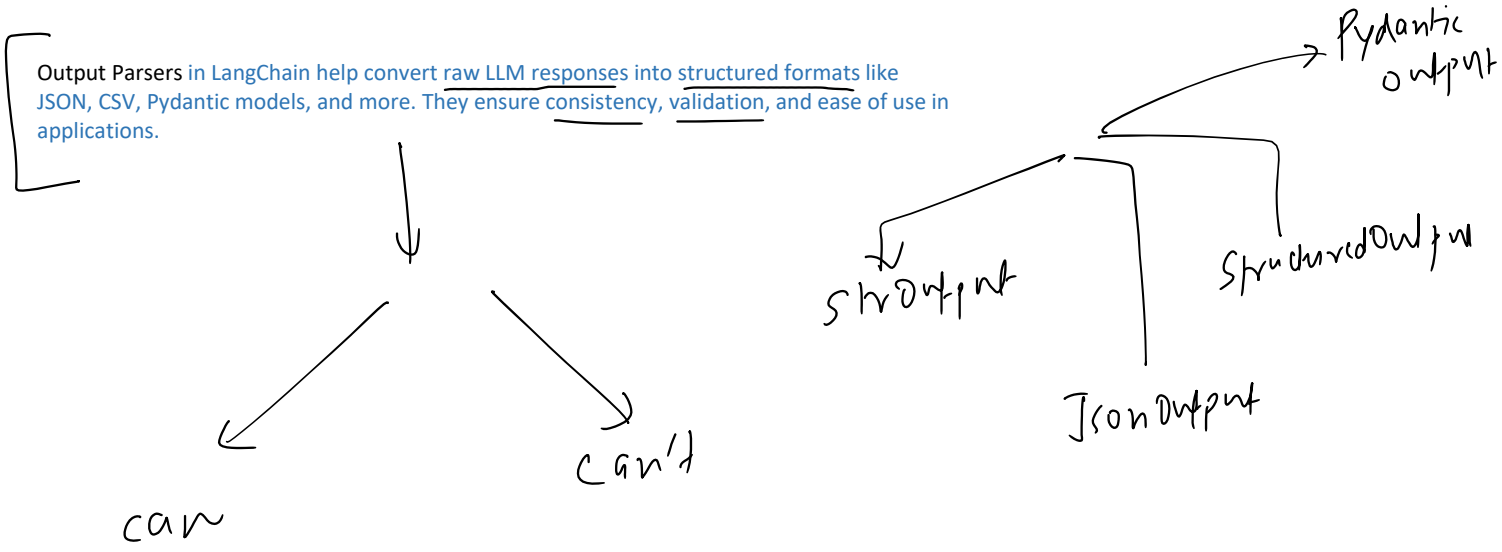
06 March 2025 16:28



Output Parsers

06 March 2025 16:29

Output Parsers in LangChain help convert raw LLM responses into structured formats like JSON, CSV, Pydantic models, and more. They ensure consistency, validation, and ease of use in applications.



StrOutputParser

06 March 2025 16:29

The StrOutputParser is the simplest output parser in LangChain. It is used to parse the output of a Language Model (LLM) and return it as a plain string.

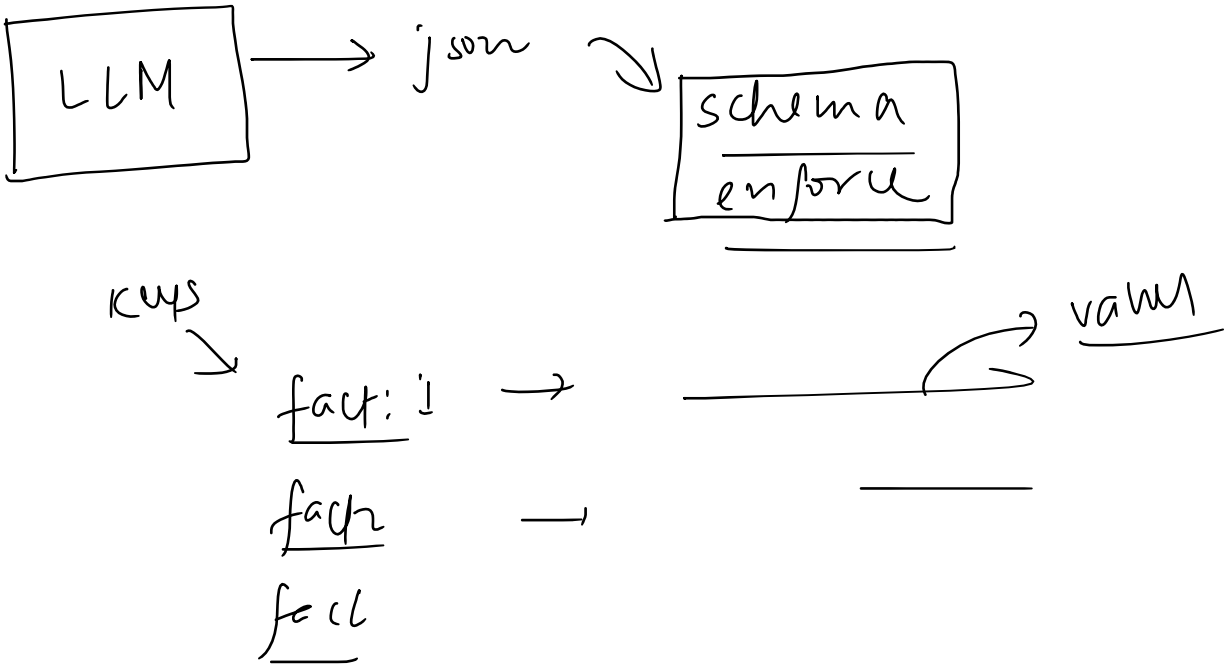
result.content

```
content='A black hole is a region in space where gravity is so strong that nothing, not e
ven light, can escape its pull. It is formed when a massive star collapses upon itself.'
additional_kwargs={'refusal': None} response_metadata={'token_usage': {'completion_tokens
': 37, 'prompt_tokens': 15, 'total_tokens': 52, 'completion_tokens_details': {'accepted_p
rediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_toke
ns': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name':
 'gpt-3.5-turbo-0125', 'system_fingerprint': None, 'finish_reason': 'stop', 'logprobs': N
one} id='run-a7b90203-58f8-47c5-a01b-01184b6aec14-0' usage_metadata={'input_tokens': 15,
'output_tokens': 37, 'total_tokens': 52, 'input_token_details': {'audio': 0, 'cache_read'
: 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}}
```

topic → LLM → [Detailed report] → LLM → 5 line summary

JSONOutputParser

06 March 2025 16:29

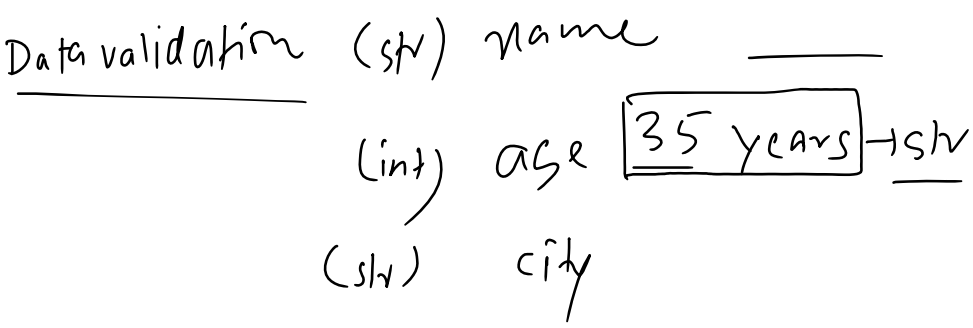


StructuredOutputParser

06 March 2025 16:29

StructuredOutputParser is an output parser in LangChain that helps extract structured JSON data from LLM responses based on predefined field schemas.

It works by defining a list of fields (ResponseSchema) that the model should return, ensuring the output follows a structured format.



PydanticOutputParser

06 March 2025 16:30

🔹 What is `PydanticOutputParser` in LangChain?

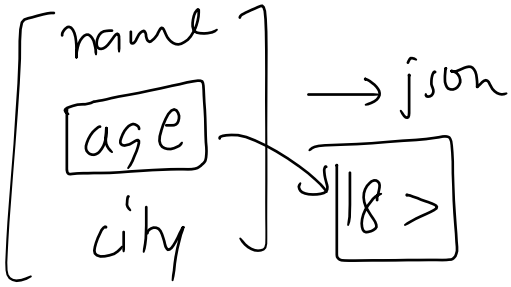
`PydanticOutputParser` is a **structured output parser** in LangChain that uses **Pydantic models** to enforce schema validation when processing LLM responses.

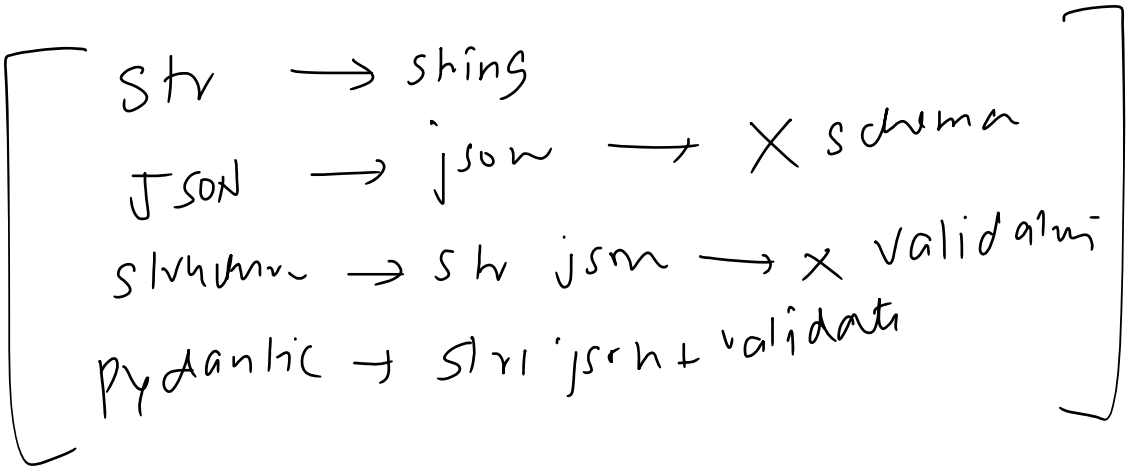
→ schema

[pydantic object]

🚀 Why Use `PydanticOutputParser`?

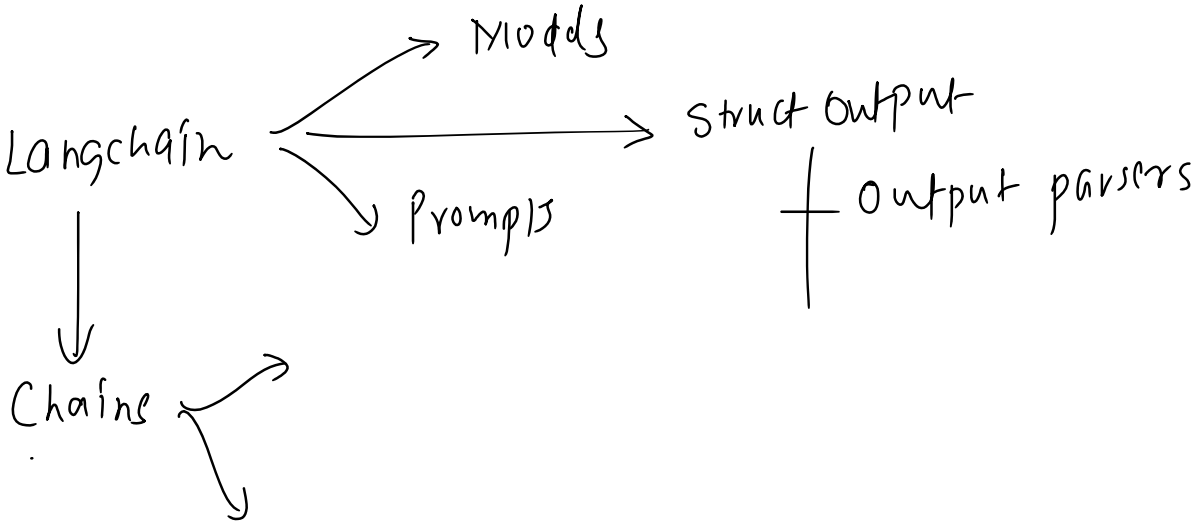
- ✅ **Strict Schema Enforcement** → Ensures that LLM responses follow a well-defined structure.
- ✅ **Type Safety** → Automatically converts LLM outputs into Python objects.
- ✅ **Easy Validation** → Uses Pydantic's built-in validation to catch incorrect or missing data.
- ✅ **Seamless Integration** → Works well with other LangChain components.





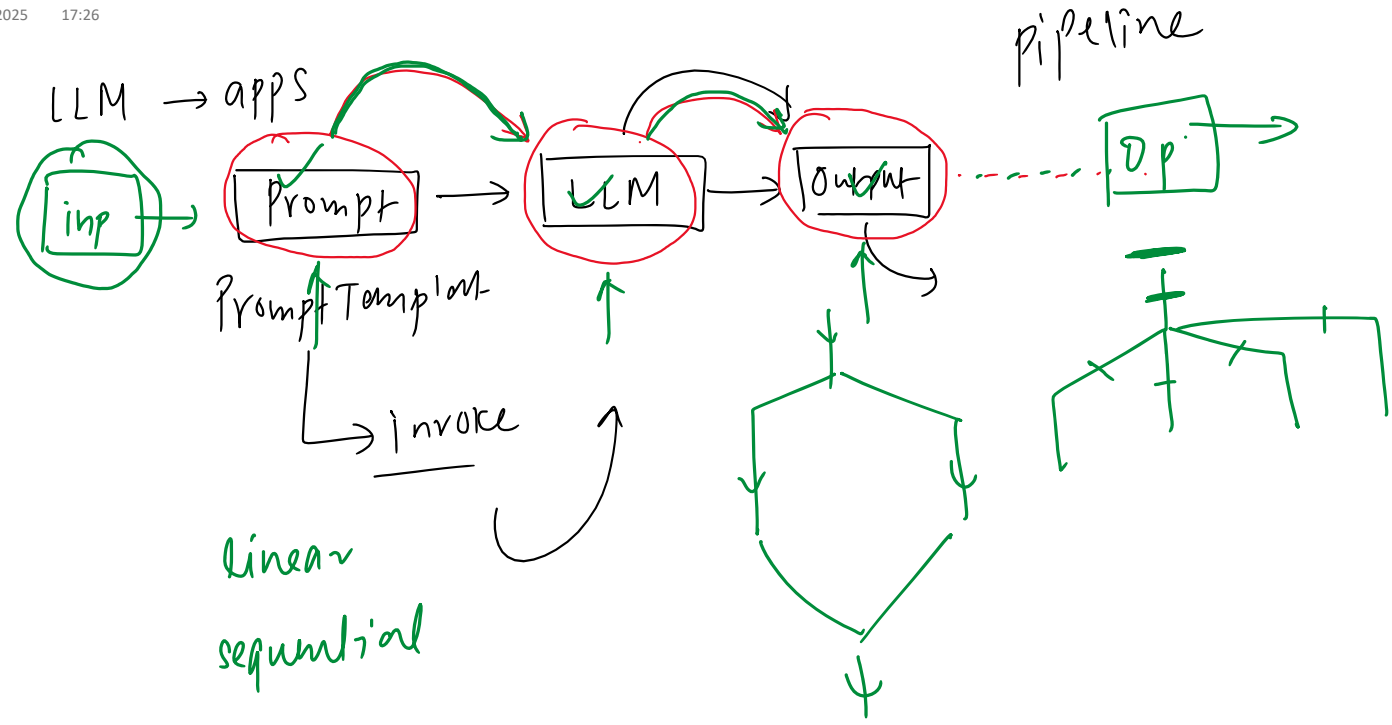
Recap

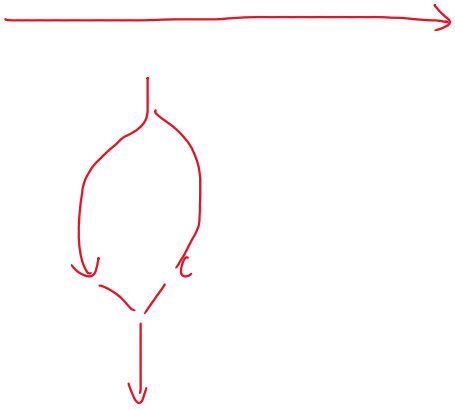
11 March 2025 17:26



What & Why

11 March 2025 17:26





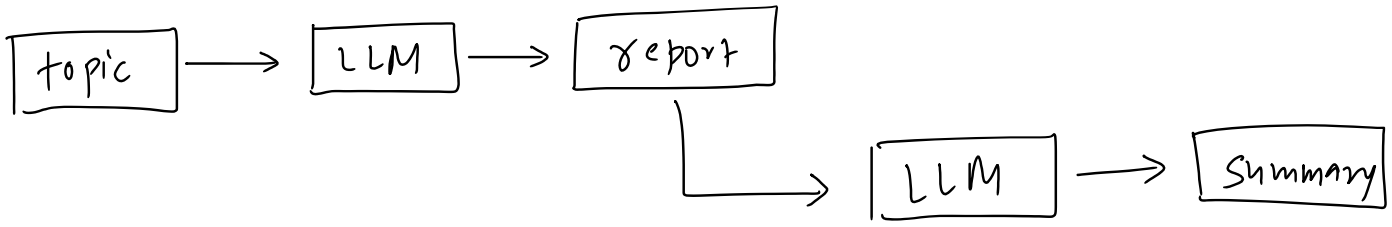
simple chains

Simple Chain

11 March 2025 17:26

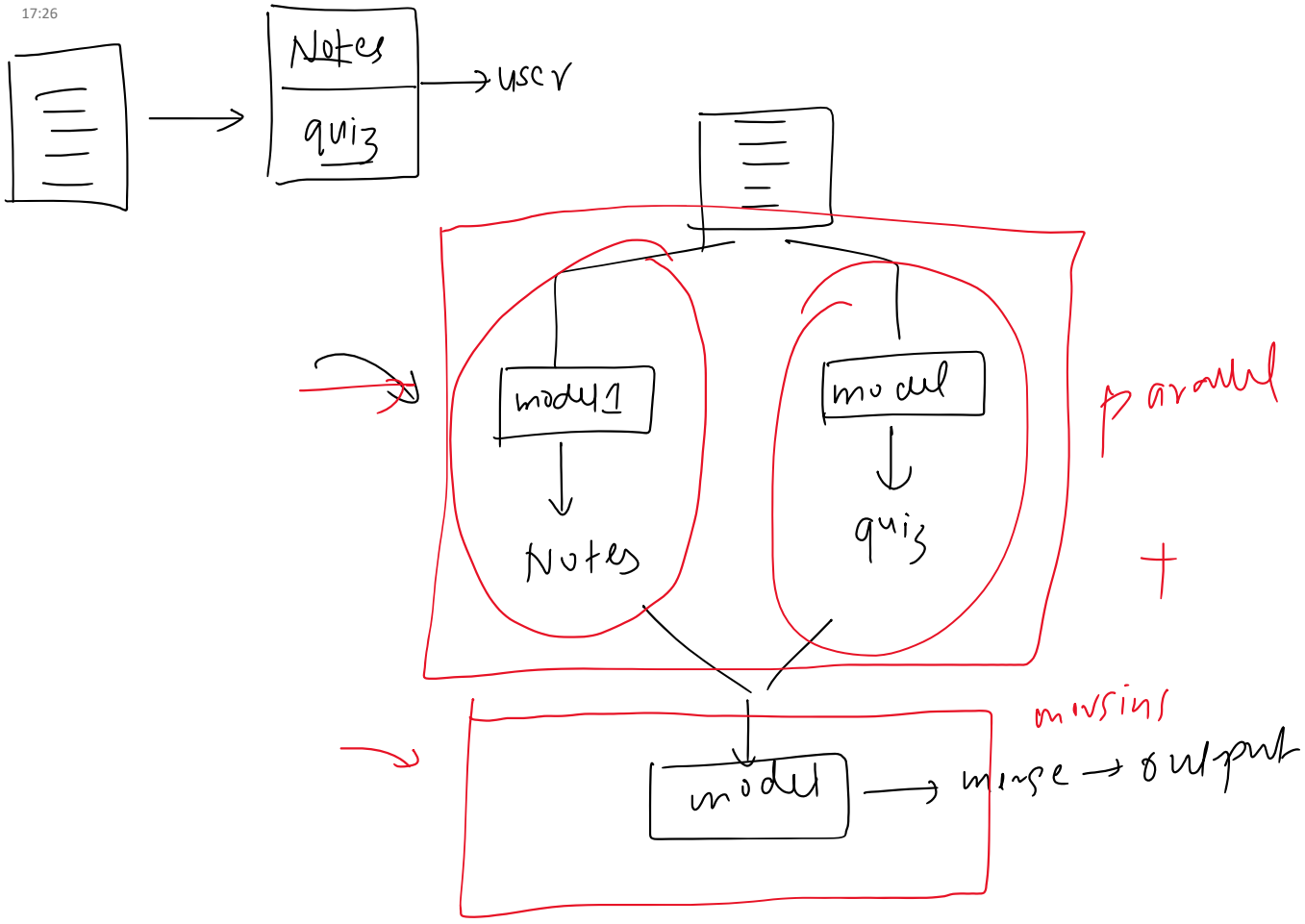
Sequential Chain

11 March 2025 17:26



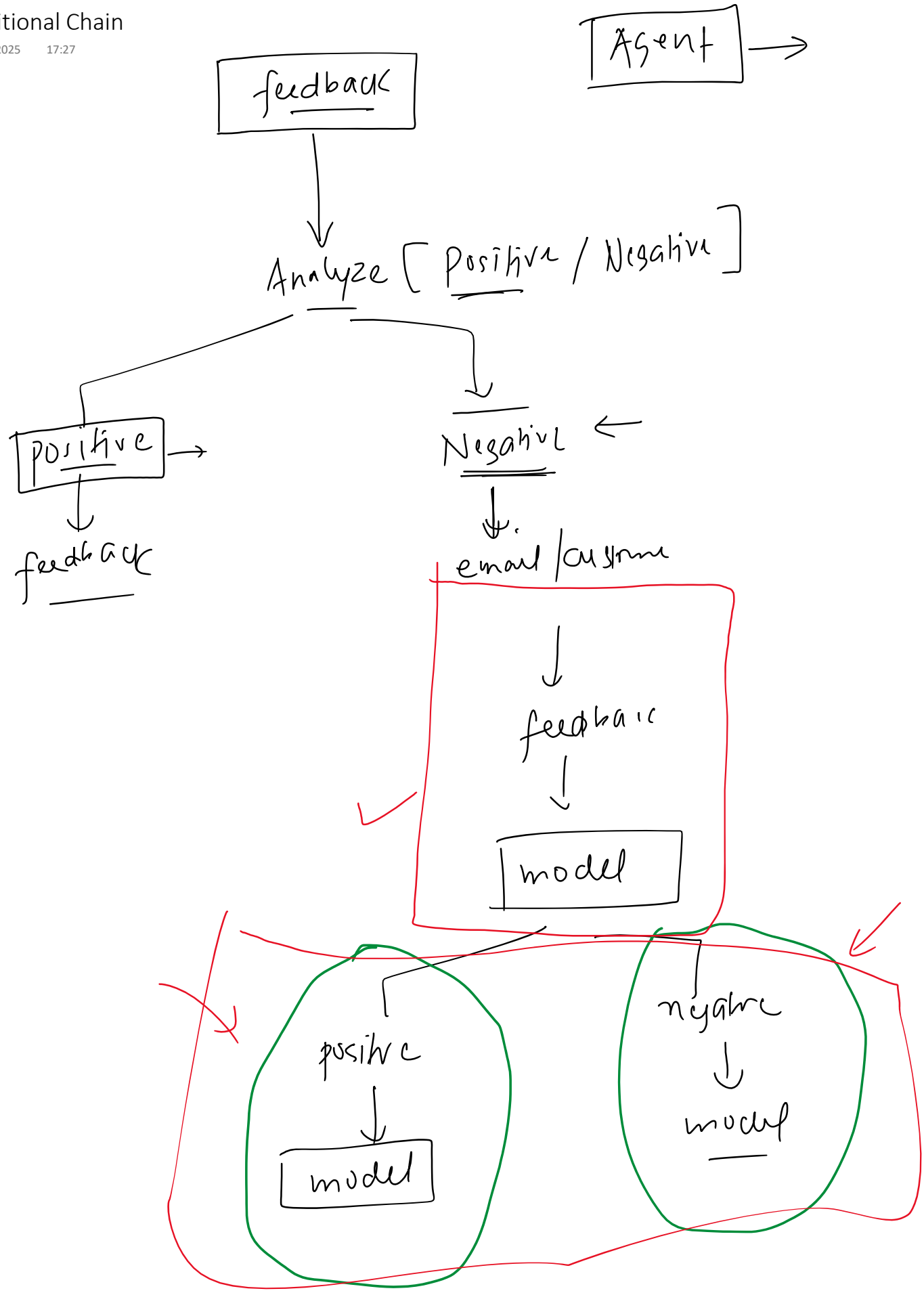
Parallel Chain

11 March 2025 17:26



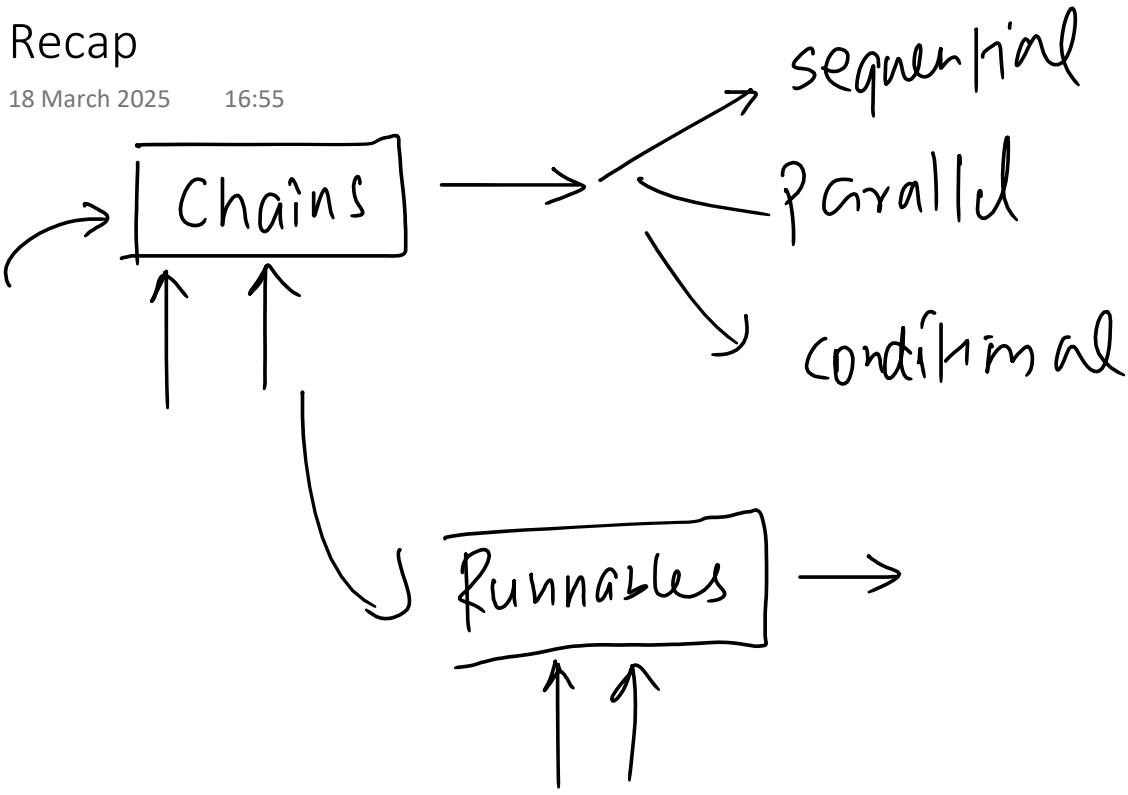
Conditional Chain

11 March 2025 17:27



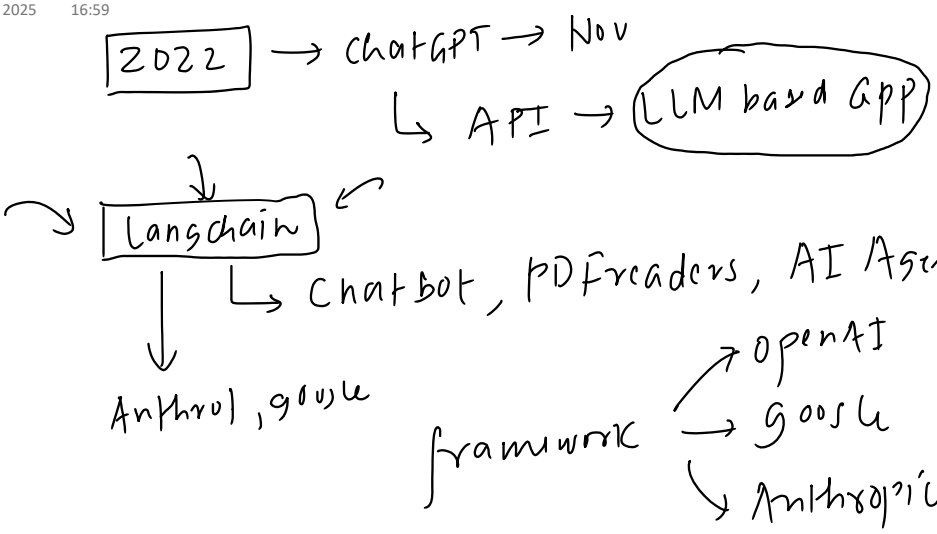
Recap

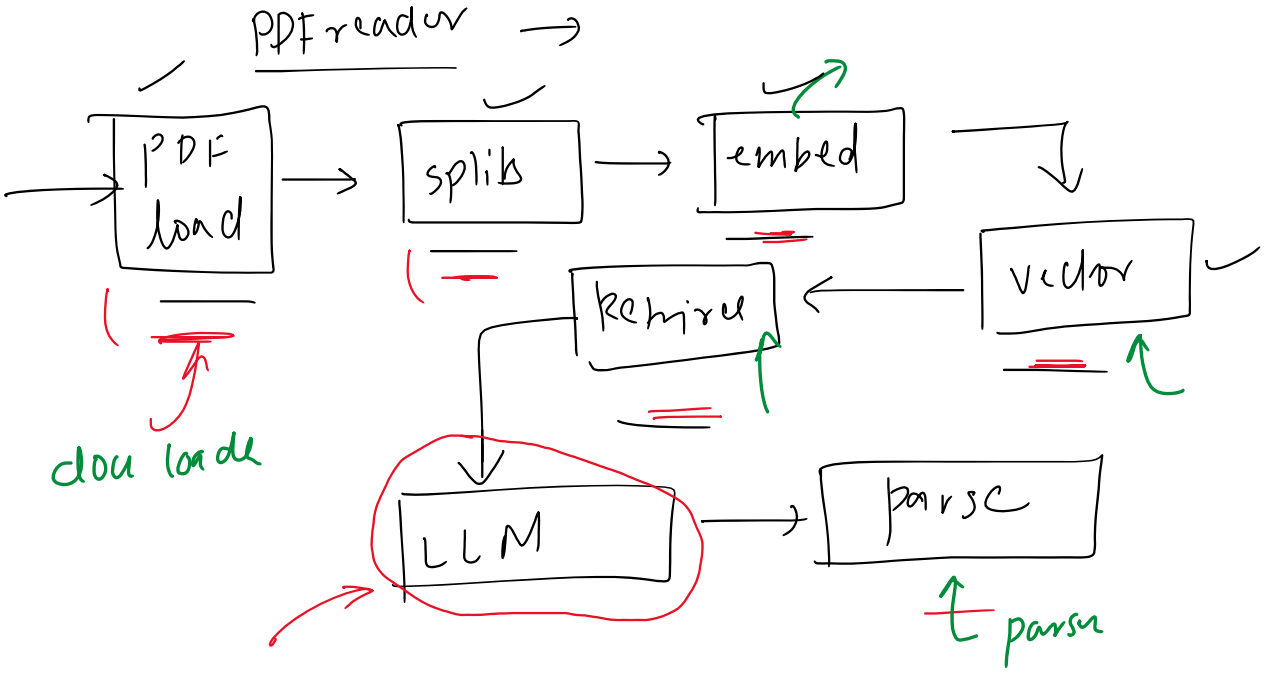
18 March 2025 16:55

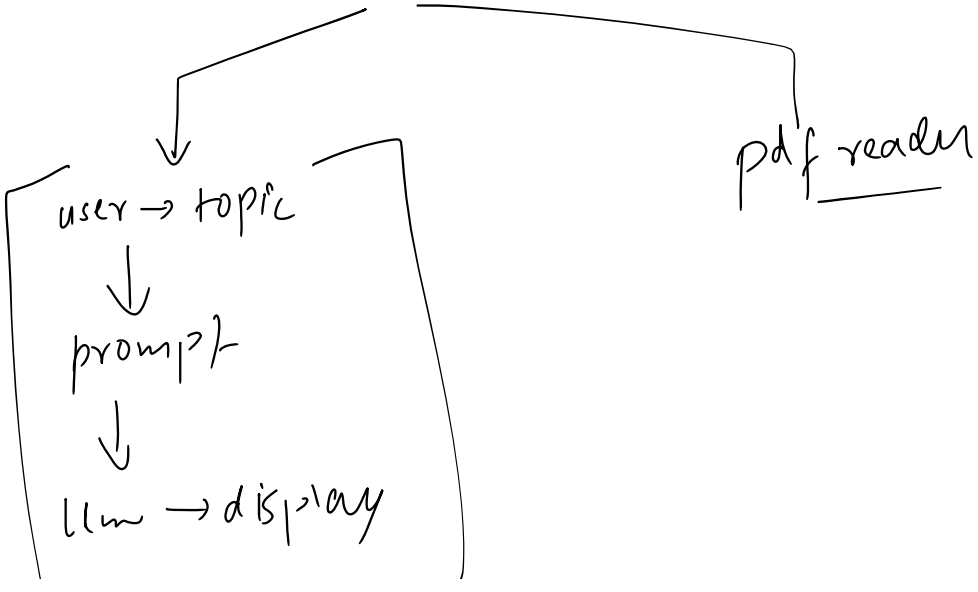


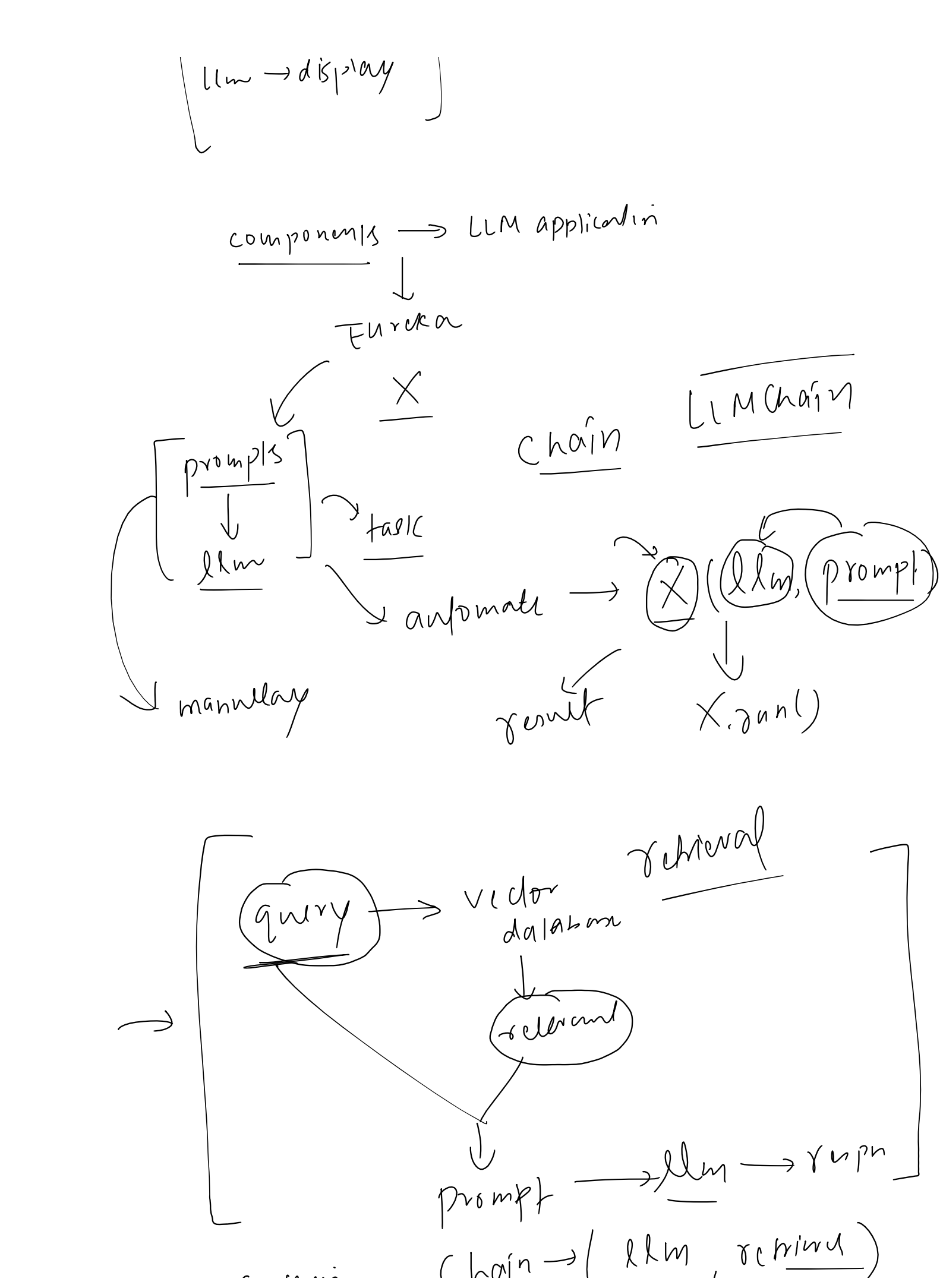
The Why

18 March 2025 16:59








llm → display
components → LLM application
Eureka
X
LLMChain
Chain
prompts
llm
task
automate
manually
X(llm, prompt)
result
X.run()
retrieval
query
vector database
relevant
prompt
llm → resp
chain → (llm , retrieval)

RetrieveQAChain Chain → (llm , retriver)

LLMChain

LLMchain ↔ LLMChain

Chain Name	Description
LLMChain	Basic chain that calls an LLM with a prompt template.
SequentialChain	Chains multiple LLM calls in a specific sequence.
SimpleSequentialChain	A simplified version of SequentialChain for easier use.
ConversationalRetrievalChain	Handles conversational Q&A with memory and retrieval.
RetrievalQA	Fetches relevant documents and uses an LLM for question-answering.
RouterChain	Directs user queries to different chains based on intent.
MultiPromptChain	Uses different prompts for different user intents dynamically.
HydeChain (Hypothetical Document Embeddings)	Generates hypothetical answers to improve document retrieval.
AgentExecutorChain	Orchestrates different tools and actions dynamically using an agent.
SQLDatabaseChain	Connects to SQL databases and answers natural language queries.

Chains
↓
use cases
↓
resive task
↓
chain

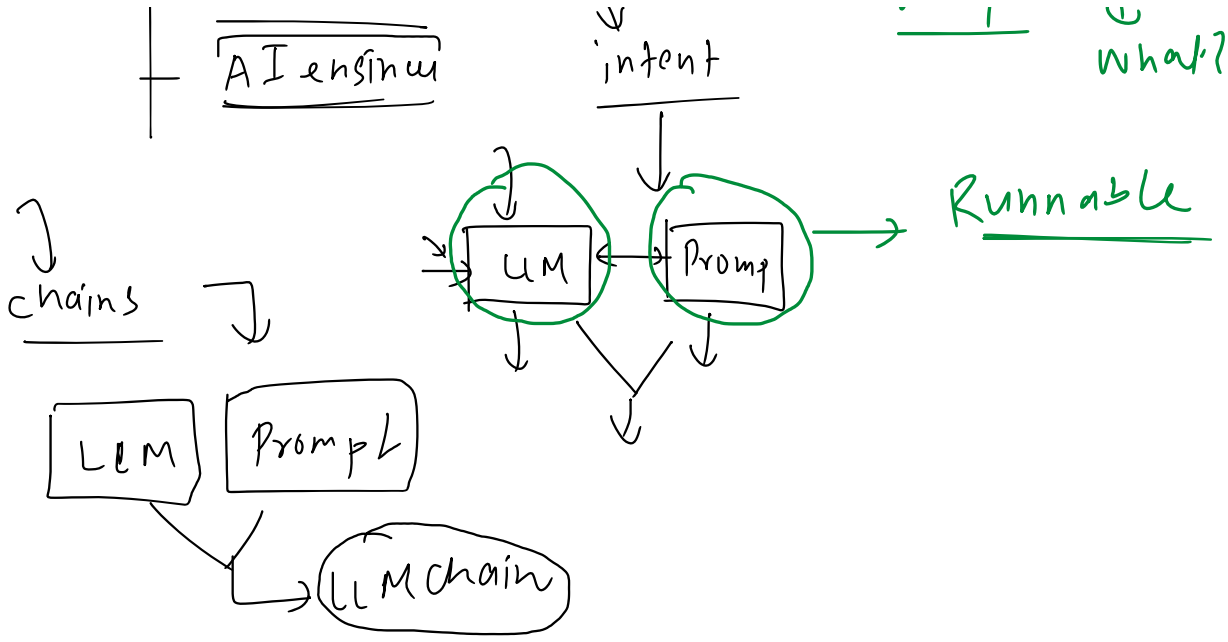
→ topic → joke → explanation

Problem

too many chains
- codebase
- AI engine

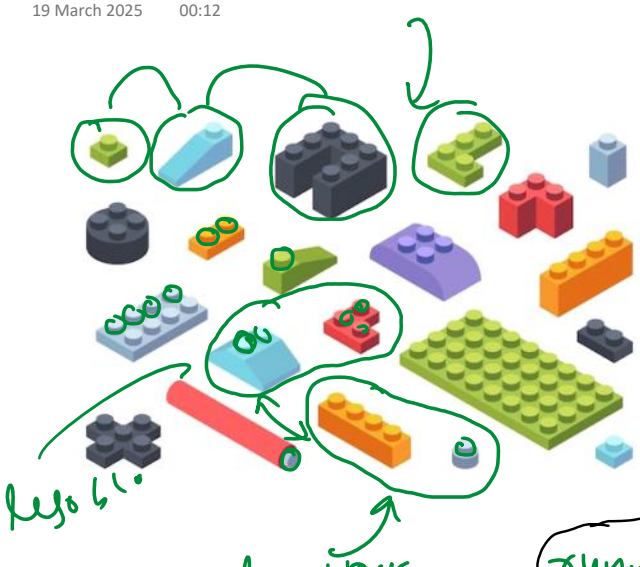
↓ intent

why? ↓ what?


AI engine
intent
what?
Runnable
LLM
Prompt
chains
LLM
Prompt
LLMchain

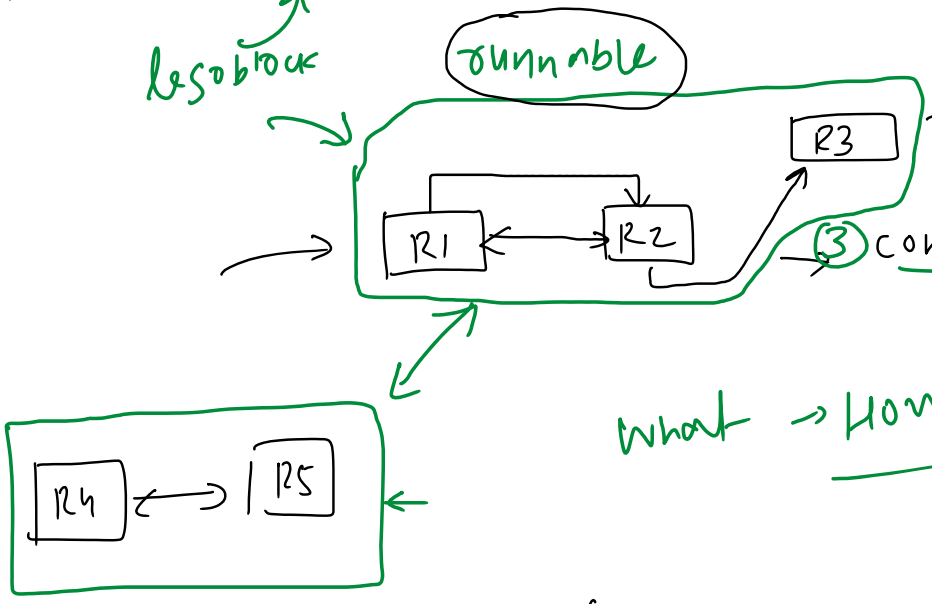
The What

19 March 2025 00:12



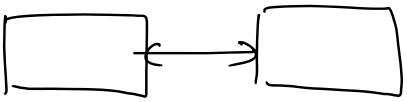
1. unit of work
   - input
   - process
   - output

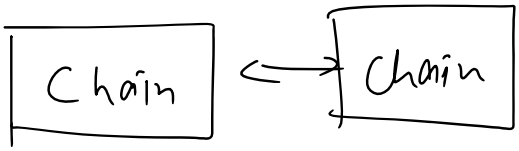
2. common interface
   - invoke()
   - batch()
   - stream()

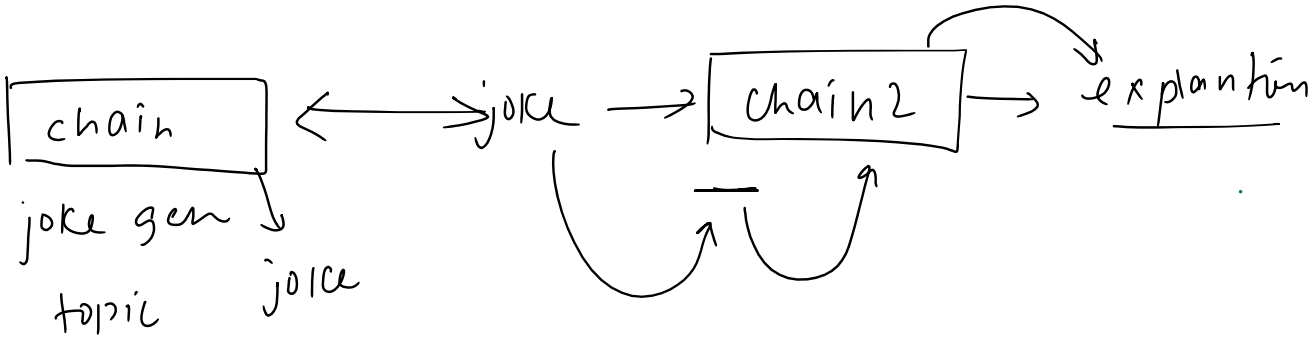


what → How →

chains

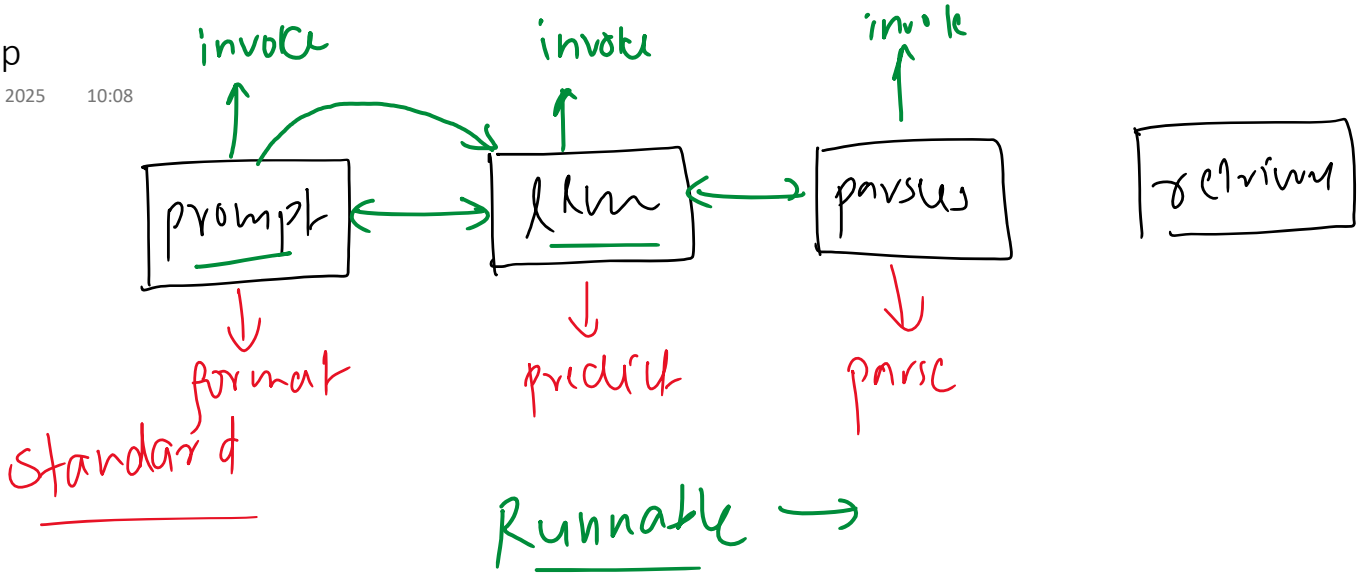






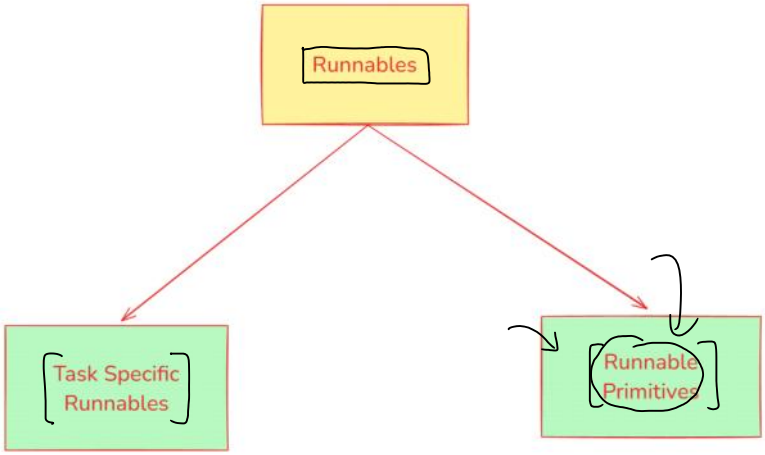
Recap

20 March 2025 10:08



Plan of Action

20 March 2025 11:02



✅ **Definition**: These are **core LangChain components** that have been converted into Runnables so they can be used in pipelines.

✅ **Purpose**: Perform **task-specific** operations like LLM calls, prompting, retrieval, etc.

✅ **Examples**:

- `ChatOpenAI` → Runs an LLM model.
- `PromptTemplate` → Formats prompts dynamically.
- `Retriever` → Retrieves relevant documents.

✅ **Definition**: These are **fundamental building blocks** for structuring execution logic in AI workflows.

✅ **Purpose**: They **help orchestrate execution** by defining how different Runnables interact (sequentially, in parallel, conditionally, etc.).

✅ **Examples**:

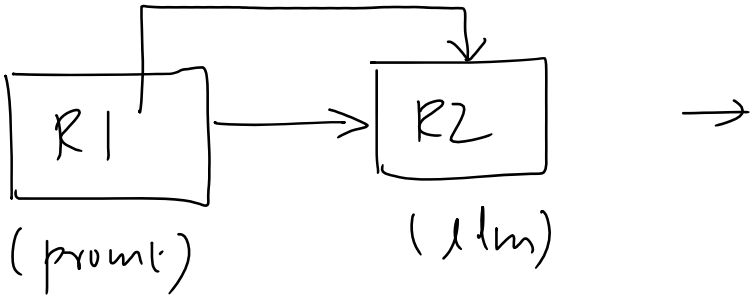
- `RunnableSequence` → Runs steps **in order** (`|` operator).
- `RunnableParallel` → Runs multiple steps **simultaneously**.
- `RunnableMap` → Maps the same input across multiple functions.
- `RunnableBranch` → Implements **conditional execution** (if-else logic).
- `RunnableLambda` → Wraps **custom Python functions** into Runnables.
- `RunnablePassthrough` → Just forwards input as output (acts as a placeholder).

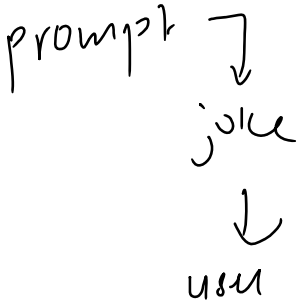
1. RunnableSequence

20 March 2025 12:10

RunnableSequence is a sequential chain of runnables in LangChain that executes each step one after another, passing the output of one step as the input to the next.

It is useful when you need to compose multiple runnables together in a structured workflow.



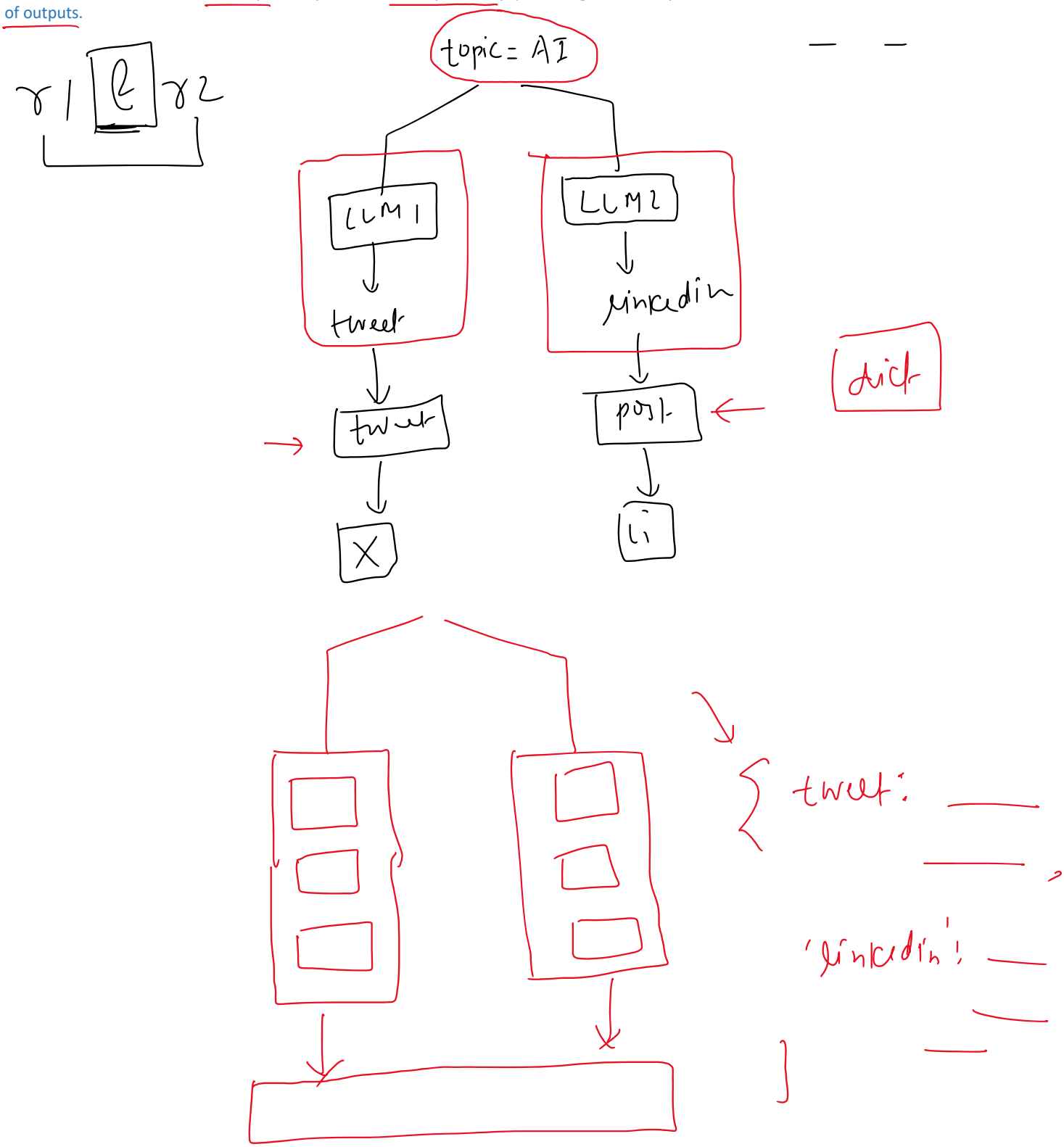


2. RunnableParallel

20 March 2025 18:33

RunnableParallel is a runnable primitive that allows multiple runnables to execute in parallel.

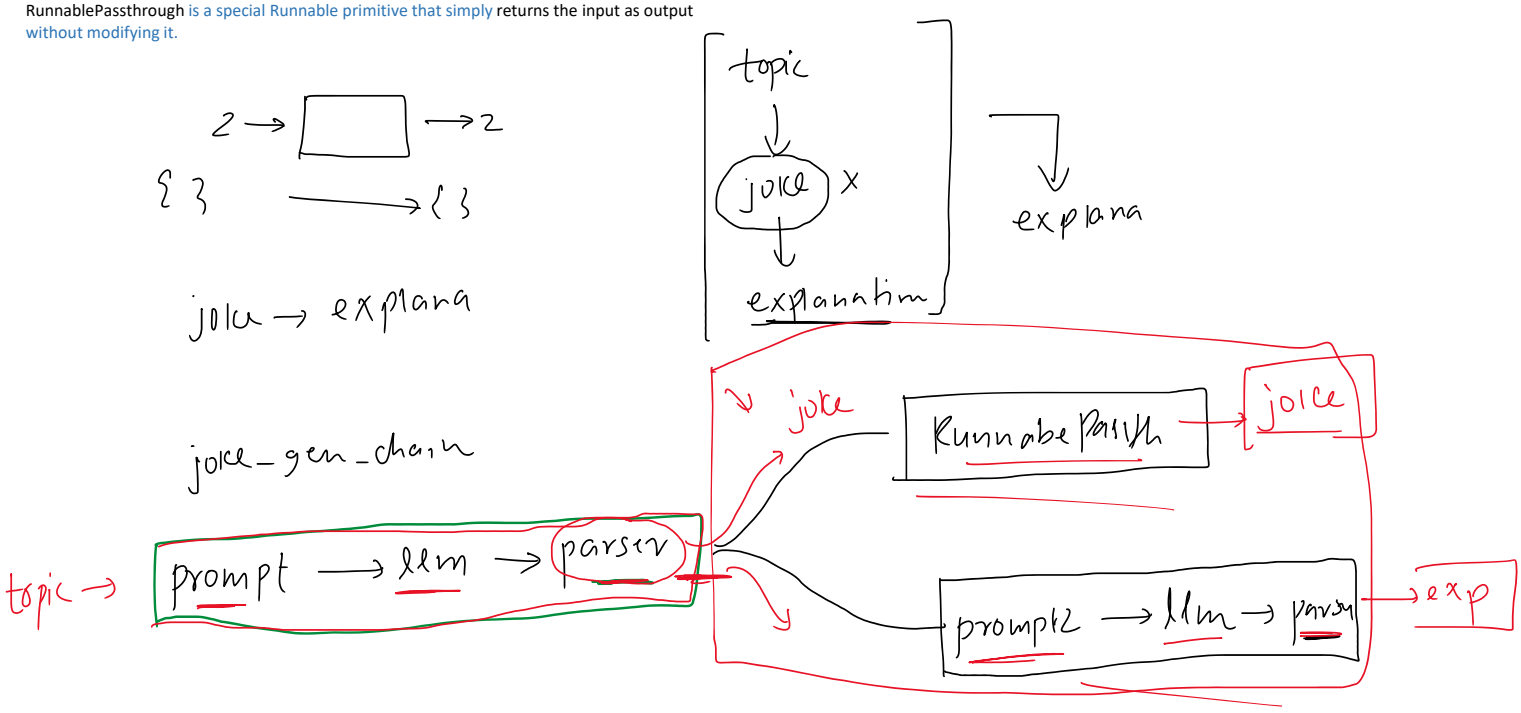
Each runnable receives the same input and processes it independently, producing a dictionary of outputs.



3. RunnablePassthrough

20 March 2025 22:34

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

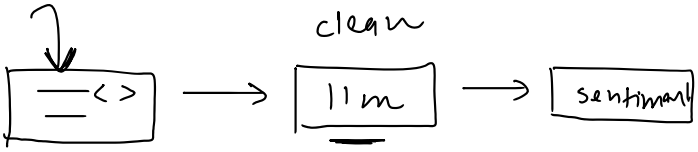


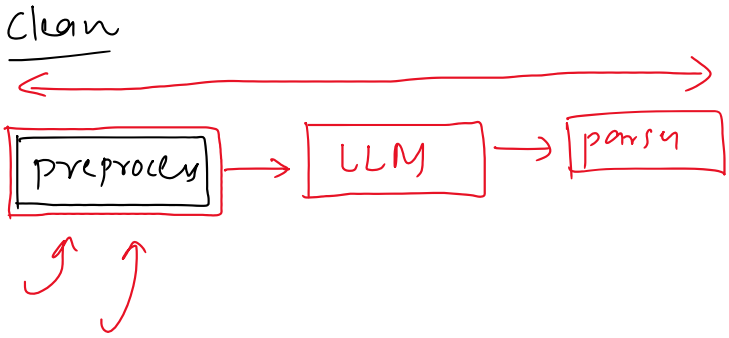
4. RunnableLambda

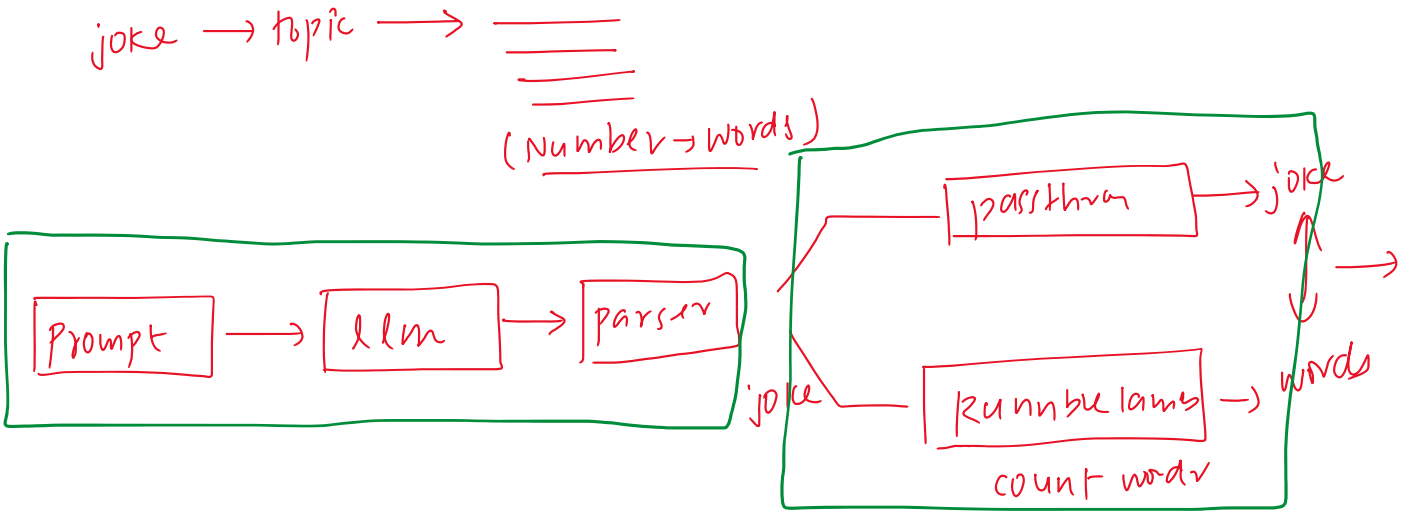
20 March 2025 23:18

RunnableLambda is a runnable primitive that allows you to apply custom Python functions within an AI pipeline.

It acts as a middleware between different AI components, enabling preprocessing, transformation, API calls, filtering, and post-processing in a LangChain workflow.





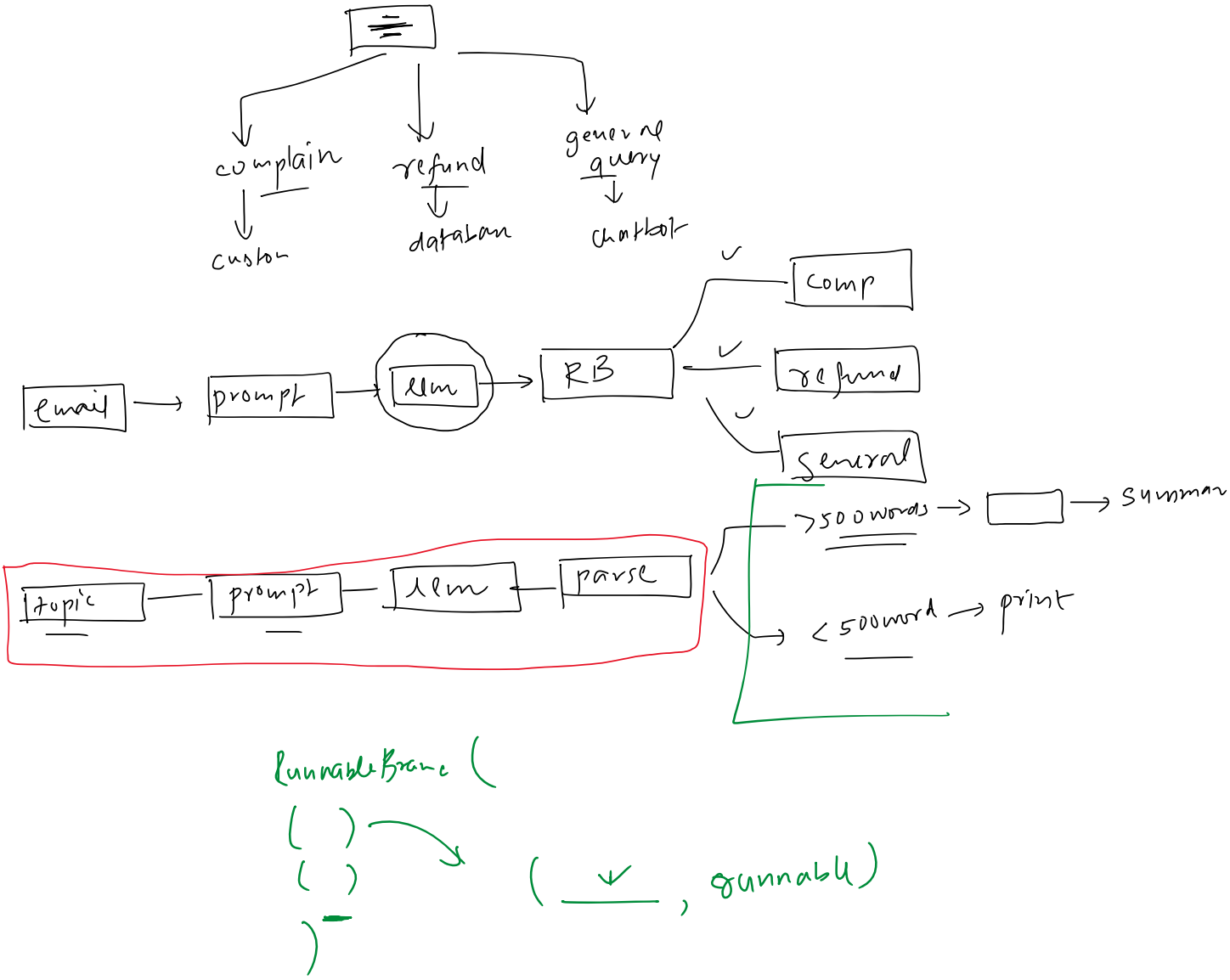


5. RunnableBranch → conditional chains

21 March 2025 08:02

RunnableBranch is a control flow component in LangChain that allows you to conditionally route input data to different chains or runnables based on custom logic.

It functions like an if/elif/else block for chains — where you define a set of condition functions, each associated with a runnable (e.g., LLM call, prompt chain, or tool). The first matching condition is executed. If no condition matches, a default runnable is used (if provided).



RunnableBranch (

( )

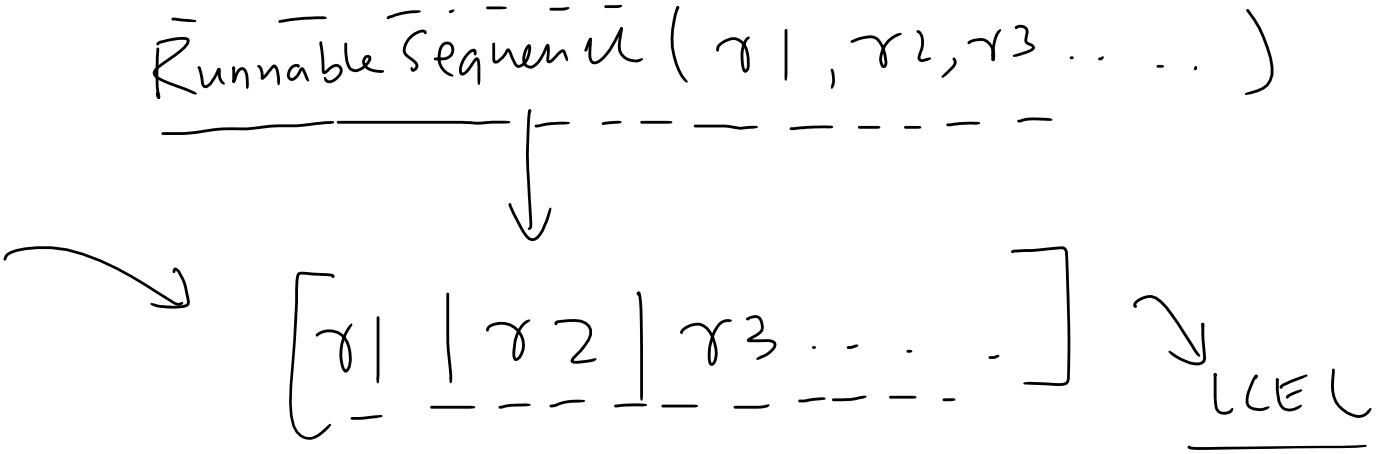
( )

)

(, runnable)

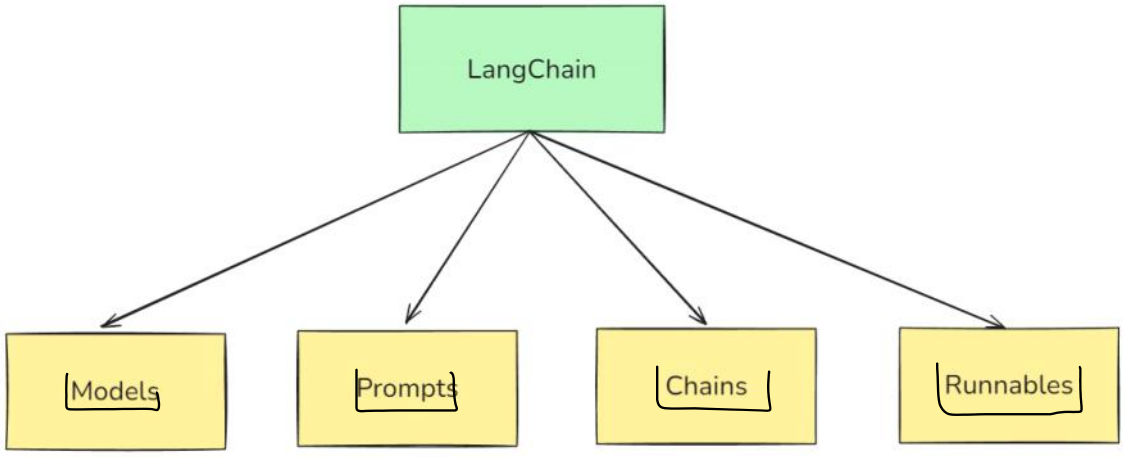
LCEL

21 March 2025 08:39



Plan of Action

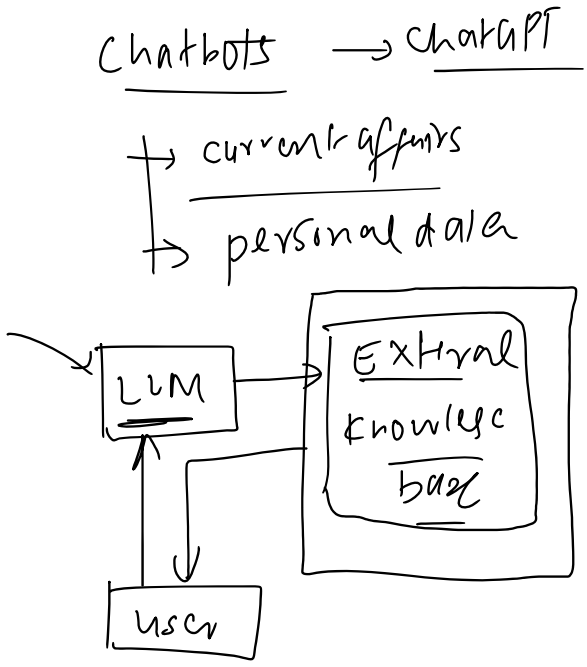
27 March 2025 10:59

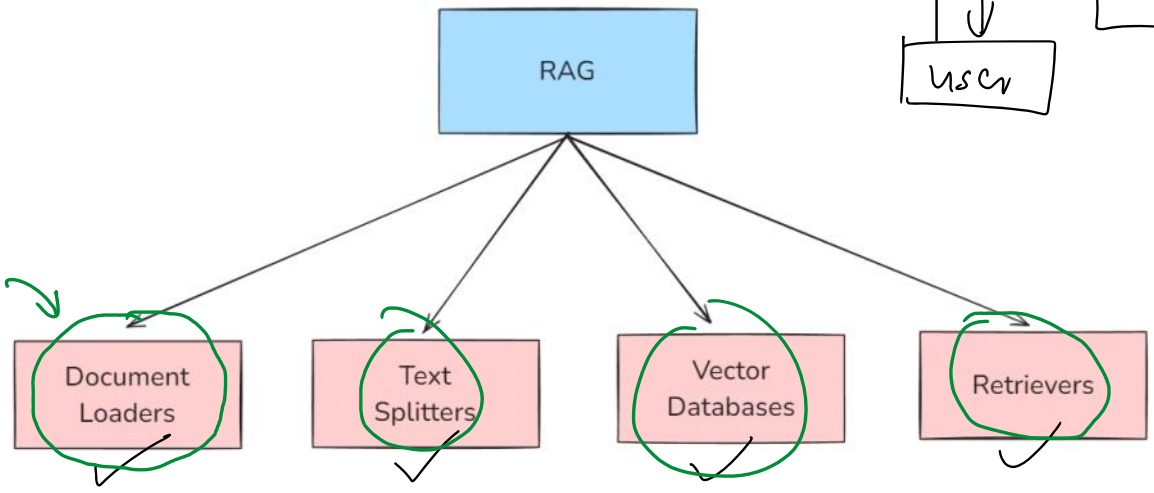


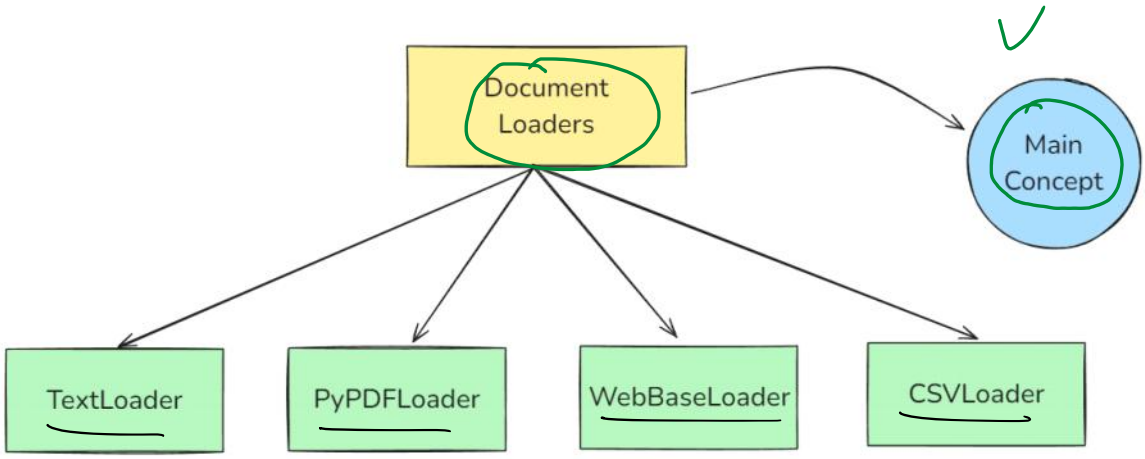
RAG is a technique that combines information retrieval with language generation, where a model retrieves relevant documents from a knowledge base and then uses them as context to generate accurate and grounded responses.

Benefits of using RAG

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





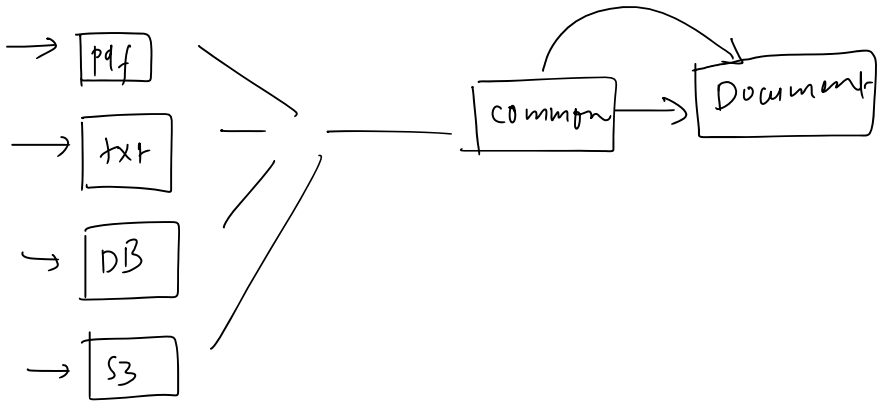

Document Loaders
Main Concept
TextLoader
PyPDFLoader
WebBaseLoader
CSVLoader

Document Loaders in LangChain

27 March 2025 16:20

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

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



TextLoader

27 March 2025 16:50

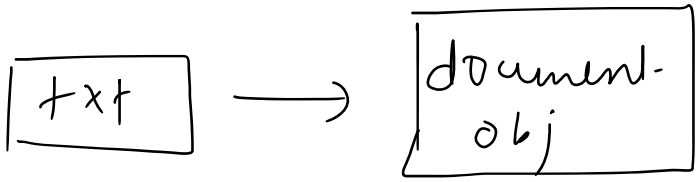
TextLoader is a simple and commonly used document loader in LangChain that reads plain text (.txt) files and converts them into LangChain Document objects.

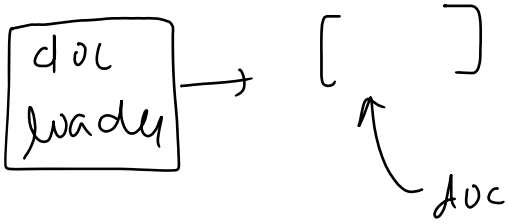
Use Case

- Ideal for loading chat logs, scraped text, transcripts, code snippets, or any plain text data into a LangChain pipeline.

Limitation

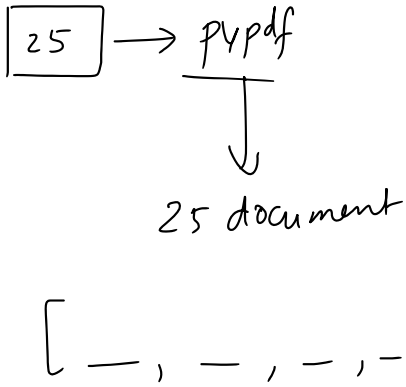
- Works only with .txt files





PyPDFLoader → text → scanned pdf

27 March 2025 17:50



PyPDFLoader is a document loader in LangChain used to load content from PDF files and convert each page into a Document object.

```
[
    Document(page_content="Text from page 1", metadata={"page": 0, "source": "file.pdf"}),
    Document(page_content="Text from page 2", metadata={"page": 1, "source": "file.pdf"}),
    ...
]
```


Limitations:

- It uses the PyPDF library under the hood — not great with scanned PDFs or complex layouts.

Use Case	Recommended Loader
Simple, clean PDFs	`PyPDFLoader`
PDFs with tables/columns	`PDFPlumberLoader`
Scanned/image PDFs	`UnstructuredPDFLoader` or `AmazonTextractPDFLoader`
Need layout and image data	`PyMuPDFLoader`
Want best structure extraction	`UnstructuredPDFLoader`

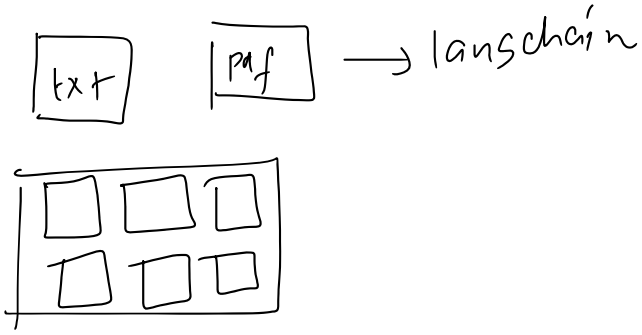
DirectoryLoader

27 March 2025 18:44

DirectoryLoader is a document loader that lets you load multiple documents from a directory (folder) of files.

Glob Pattern	What It Loads
`"**/*.txt"`	All `.txt` files in all subfolders
`"*.pdf"`	All `.pdf` files in the root directory
`"data/*.csv"`	All `.csv` files in the `data/` folder
`"**/*"`	All files (any type, all folders)

`**` = recursive search through subfolders



Load vs Lazy load → 500 pdf → generator of docs →

27 March 2025 23:51

eager loading

`load()`

- **Eager Loading** (loads everything at once).
- Returns: A **list of** `Document` **objects**.
- Loads all documents **immediately** into memory.
- Best when:
  - The number of documents is small.
  - You want everything loaded upfront.

`lazy_load()`

- **Lazy Loading** (loads on demand).
- Returns: A **generator** of `Document` objects.
- Documents are **not all loaded at once**; they're fetched one at a time as needed.
- Best when:
  - You're dealing with **large documents or lots of files**.
  - You want to **stream** processing (e.g., chunking, embedding) without using lots of memory.

WebBaseLoader

28 March 2025 00:34

WebBaseLoader is a document loader in LangChain used to load and extract text content from web pages (URLs).

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

When to Use:

- For blogs, news articles, or public websites where the content is primarily text-based and static.

Limitations:

- Doesn't handle JavaScript-heavy pages well (use SeleniumURLLoader for that).
- Loads only static content (what's in the HTML, not what loads after the page renders).

CSVLoader

28 March 2025 01:48

CSVLoader is a document loader used to load CSV files into LangChain Document objects — one per row, by default.

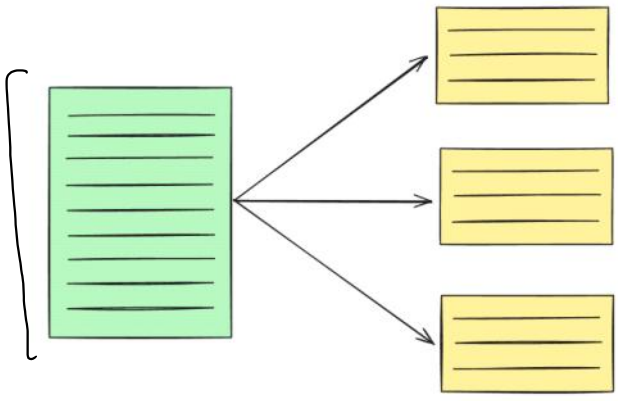
Other Document Loaders

28 March 2025 01:58

Text Splitting

01 April 2025 18:10

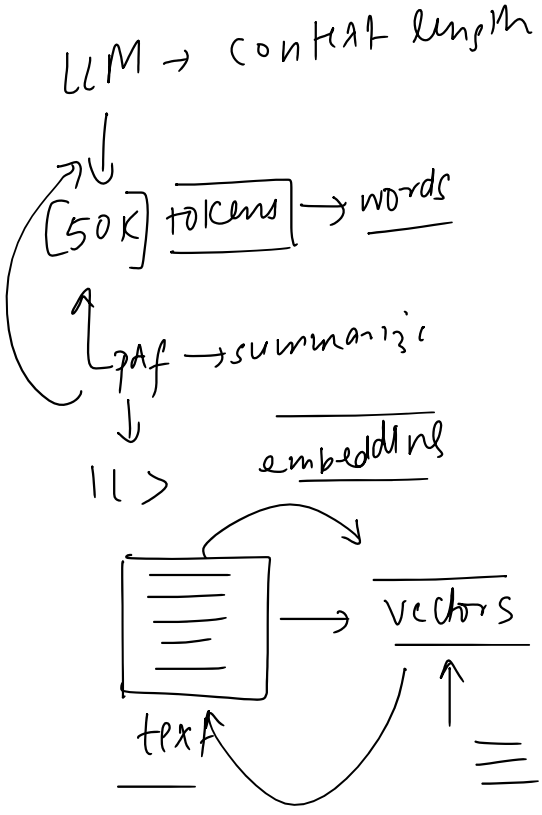
Text Splitting is the process of breaking large chunks of text (like articles, PDFs, HTML pages, or books) into smaller, manageable pieces (chunks) that an LLM can handle effectively.

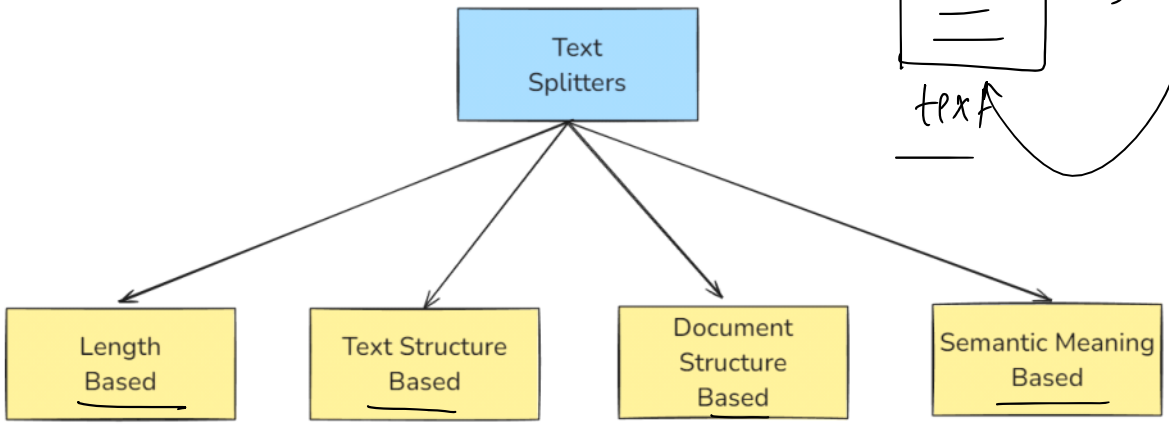


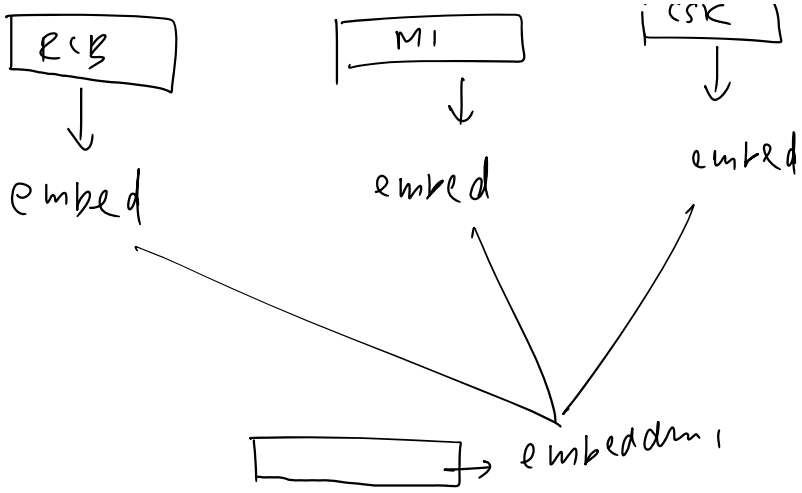
- Overcoming model limitations: Many embedding models and language models have maximum input size constraints. Splitting allows us to process documents that would otherwise exceed these limits.
- Downstream tasks - Text Splitting improves nearly every LLM powered task

Task	Why Splitting Helps
Embedding	Short chunks yield more accurate vectors
Semantic Search	Search results point to focused info, not noise
Summarization	Prevents hallucination and topic drift

- Optimizing computational resources: Working with smaller chunks of text can be more memory-efficient and allow for better parallelization of processing tasks.





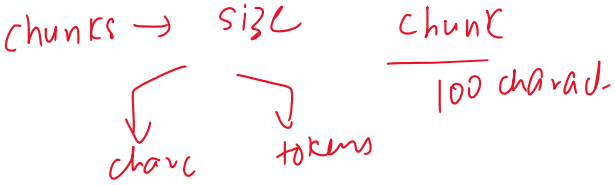

RCB
MI
CSK
embed
embed
embed
embeddings

1. Length Based Text Splitting

01 April 2025 18:10

Space exploration has led to incredible scientific discoveries. From landing on the Moon to exploring Mars, humanity continues to push the boundaries of what's possible beyond our planet.

These missions have not only expanded our knowledge of the universe but have also contributed to advancements in technology here on Earth. Satellite communications, GPS, and even certain medical imaging techniques trace their roots back to innovations driven by space programs.



Space exploration has led to incredible scientific discoveries. From landing on the Moon to explorin → c1

g Mars, humanity continues to push the boundaries of what's possible beyond our planet. These missi c2

ons have not only expanded our knowledge of the universe but have also contributed to advancements in c3

n technology here on Earth. Satellite communications, GPS, and even certain medical imaging techniqu c4

es trace their roots back to innovations driven by space programs. c5

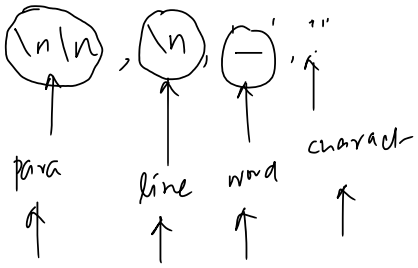
2. Text-Structured Based

01 April 2025 18:10

My name is Nitish
I am 35 years old

I live in Gurgaon
How are you

structure



chunks

Allowed

[chunk_size = 10]

chunksize = 10

[My name is Nit

My name is Nitish (17)
I am 35 years old (17)
I live in Gurgaon (17)
How are you (11)

My name is Nitish (17)/X
I am 35 years old (17)
(34)

I live in Gurgaon (17)
How are you (11)
(28) X

My name is Nitish (17)

My (2) name (4) is (2) Nitish (6)

My name (7) is (2) Nitish (6)

My name is (10) Nitish (6)

[My name is] [Nitish]

I am 35 years old (17)

I (1) am (2) 35 (2) years (5) old (3)

[I am 35] (7) [years old]

I live in Gurgaon (17)

I (1) live (4) in (2) gurgaon (7)

[I live in] [gurgaon]

How are you (11)

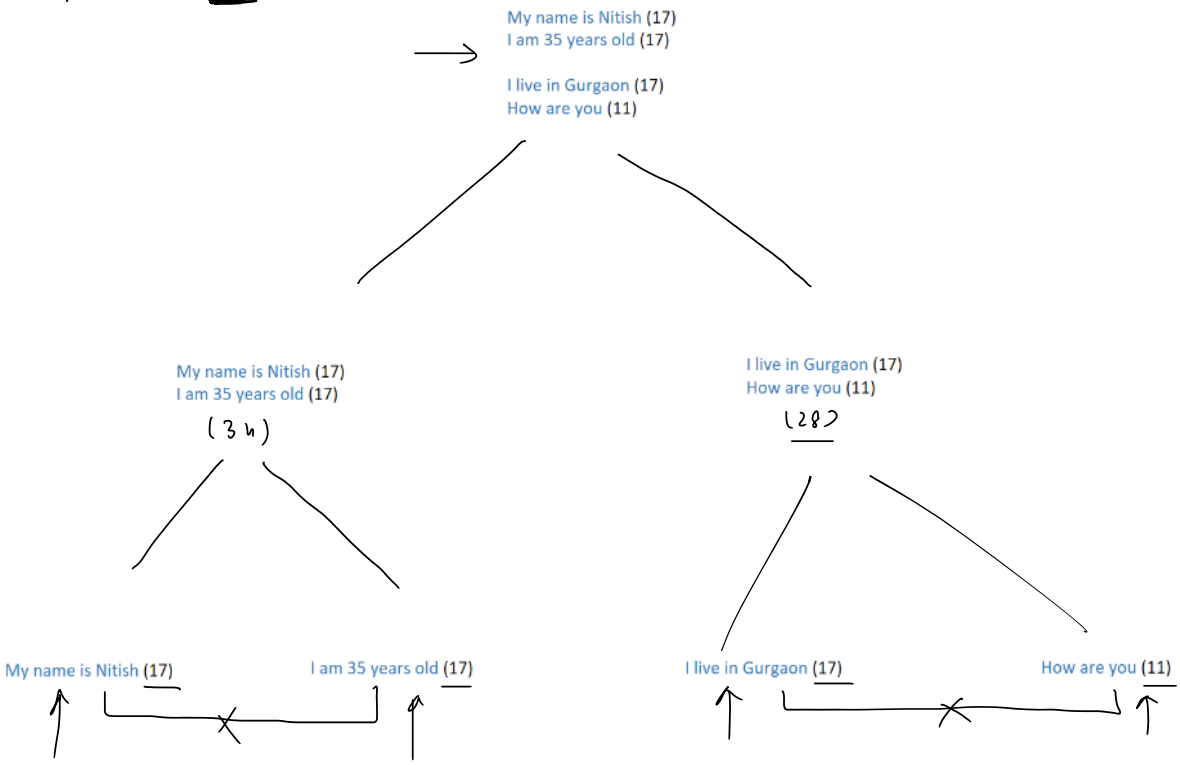
How (3) are (3) you (3)

[How are] [you

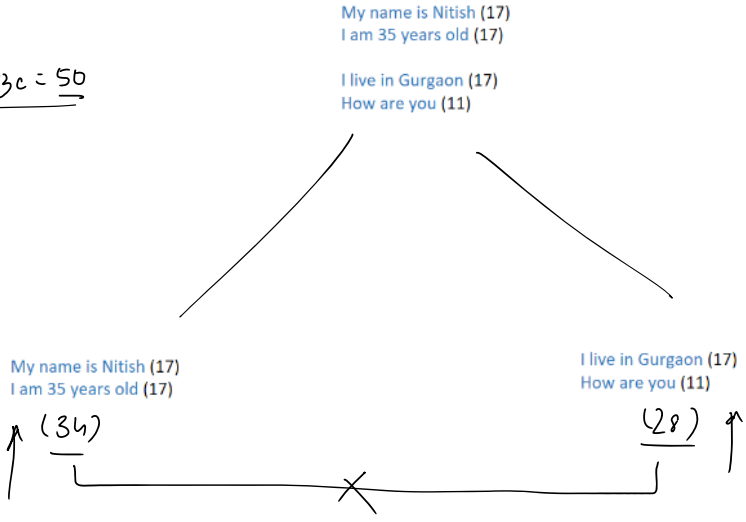
Chunk size = 25

My name is Nitish (17)
I am 35 years old (17)

Chunk size = 25



Chunk size = 50



3. Document-Structured Based

01 April 2025 18:11

markdown → LLM

```
# 📘 Project Name: Smart Student Tracker

A simple Python-based project to manage and track student data,

---

## 🔍 Features

- Add new students with relevant info
- View student details
- Check if a student is passing
- Easily extendable class-based design

---

## 🛠 Tech Stack

- Python 3.10+
- No external dependencies
```

code →

```python
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade  # Grade is a float (like 8.5 or 9.2)

    def get_details(self):
        return f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}"

    def is_passing(self):
        return self.grade >= 6.0


# Example usage
student1 = Student("Aarav", 20, 8.2)
print(student1.get_details())

if student1.is_passing():
    print("The student is passing.")
else:
    print("The student is not passing.")
```

```
# First, try to split along Markdown headings (starting with level 2)
"\n#{1,6} ",
# Note the alternative syntax for headings (below) is not handled here
# Heading level 2
# ---------------
# End of code block
"```\n",
# Horizontal lines
"\n\\*\\*\\*+\n",
"\n---+\n",
"\n___+\n",
# Note that this splitter doesn't handle horizontal lines defined
# by *three or more* of ***, ---, or ___, but this is not handled
"\n\n",
"\n",
" ",
"",
```

```
# First, try to split along class definitions
"\nclass ",
"\ndef ",
"\n\tdef ",
# Now split by the normal type of lines
"\n\n",
"\n",
" ",
"",
```

4. Semantic Meaning Based

01 April 2025 18:11

2 chunks

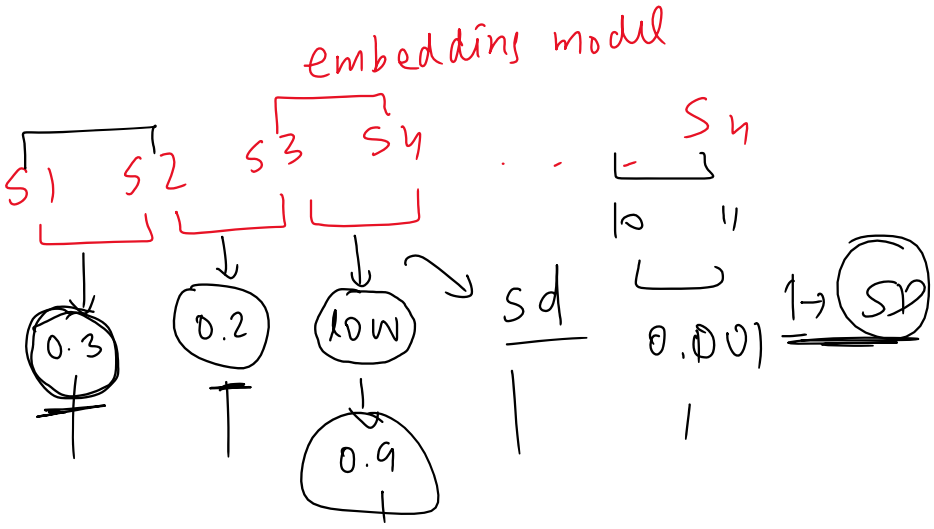
(len) (structure)

Farmers were working hard in the fields, preparing the soil and planting seeds for the next season. The sun was bright, and the air smelled of earth and fresh grass.

The Indian Premier League (IPL) is the biggest cricket league in the world. People all over the world watch the matches and cheer for their favourite teams.

→ 2 para ✗

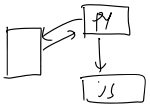
Terrorism is a big danger to peace and safety. It causes harm to people and creates fear in cities and villages. When such attacks happen, they leave behind pain and sadness. To fight terrorism, we need strong laws, alert security forces, and support from people who care about peace and safety.



Why Vector Stores?

05 April 2025 17:40

Movie id	Movie name	Director	Actor	Genre	Release Date	Outcome
M001	3 Idiots	Raju Hirani	Aamir Khan	Drama, Romance	2009	Super Hit
M002	Chennai Express	Rohit Shetty	Shah Rukh Khan	Romance, Comedy	2014	Super Hit
M003	Inception	C Nolan	L Di Caprio	Thriller, Sci-Fi	2009	Blockbuster
M004	Stree	Amar Kaushik	Rajkumar Rao	Horror, Comedy	2019	Hit



Keyword match

m1 m2

Movie id	Plot
M001	In the present day, Farhan receives a call from Chatur, saying that Ranchi is coming. Farhan is so excited that he fakes a heart attack to get off a flight and picks up Raju from his home (who forgets to wear his pants). Farhan and Raju meet Chatur at the water tower of their college ICE (Imperial College of Engineering), where Chatur informs them that he has found Rancho. Chatur taunts Farhan and Raju that now he has a mansion worth $3.5 million in the US, with a heated swimming pool, a maple wood floor living room and a Lamborghini for a car. Chatur reveals that Rancho is in Shimla...
M002	Rahul Mithaiwala (Shahrukh Khan) is a forty-year old bachelor who lives in Mumbai. His parents died in a car accident when he was eight years old and was brought up by grandparents. His grandfather has a sweet-selling chain store - Y.Y. Mithaiwala. Before his birth centenary celebration, two of Rahul's friends suggest a vacation in Goa which he accepts. On the eve of the celebration, his grandfather dies whilst watching a cricket match. His grandmother tells him that his grandfather desired to have his ashes divided into two parts - to be immersed in the Ganges River and Rameswaram respectively. She requests Rahul to go to Rameswaram and immerse them. Rahul reluctantly accepts her request but was also eager to attend the Goa trip...
M003	Dominick "Dom" Cobb (Leonardo DiCaprio) and business partner Arthur (Joseph Gordon-Levitt) are "extractors", people who perform corporate espionage using an experimental military technology to infiltrate the subconscious of their targets and extract information while experiencing shared dreaming. Their latest target is Japanese businessman Saito (Ken Watanabe). The extraction from Saito fails when sabotaged by a memory of Cobb's deceased wife Mal (Marion Cotillard). After Cobb's and Arthur's associate sells them out, Saito reveals that he was actually auditioning the team to perform the difficult act of "inception": planting an idea in a person's subconscious...
M004	In the peculiar town of Chanderi, India, the residents believe in the myth of an angry woman ghost, referred to as "Stree" (Hindi for woman) (Flora Saini), who stalks men during Durga Puja festival. This is explained by the sudden disappearance of these men, leaving their clothes behind. She is said to stalk the men of the town, whispering their names and causing disappearances if they look back at her. The whole town protects itself from Stree during the 4 nights of Durga Puja by writing "OO Stree, Kal Aana" on their walls. Additionally, men are advised to avoid going out alone after 10 PM during the festival and to move in groups for safety. This practice reflects a societal parallel to the precautions typically advised to women for their own protection...

512 dim

0.32	0.63	0.13	0.45	0.0	0.1	•	•	•	0.78	0.99

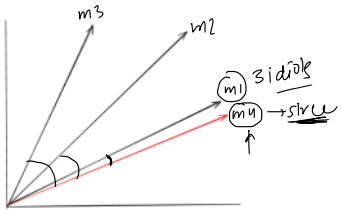
compare similarity score →

0.44	0.61	0.0	0.91	0.0	0.99	•	•	•	0.91	0.99

0.21	0.0	0.0	0.01	0.0	0.77	•	•	•	0.0	0.99

0.33	0.03	0.10	0.22	0.99	0.77	•	•	•	0.22	0.88

512 dim

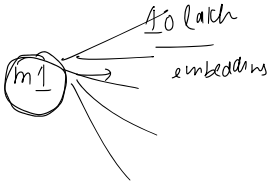


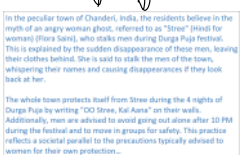
C1 → gen embedding vectors

C2 → storage

C3 → semantic search intelligence

↳ Vector store





vectors

0.33	0.03	0.10	0.22	0.99	0.77	•	•	•	0.22	0.88

What are Vector Stores

05 April 2025 17:38

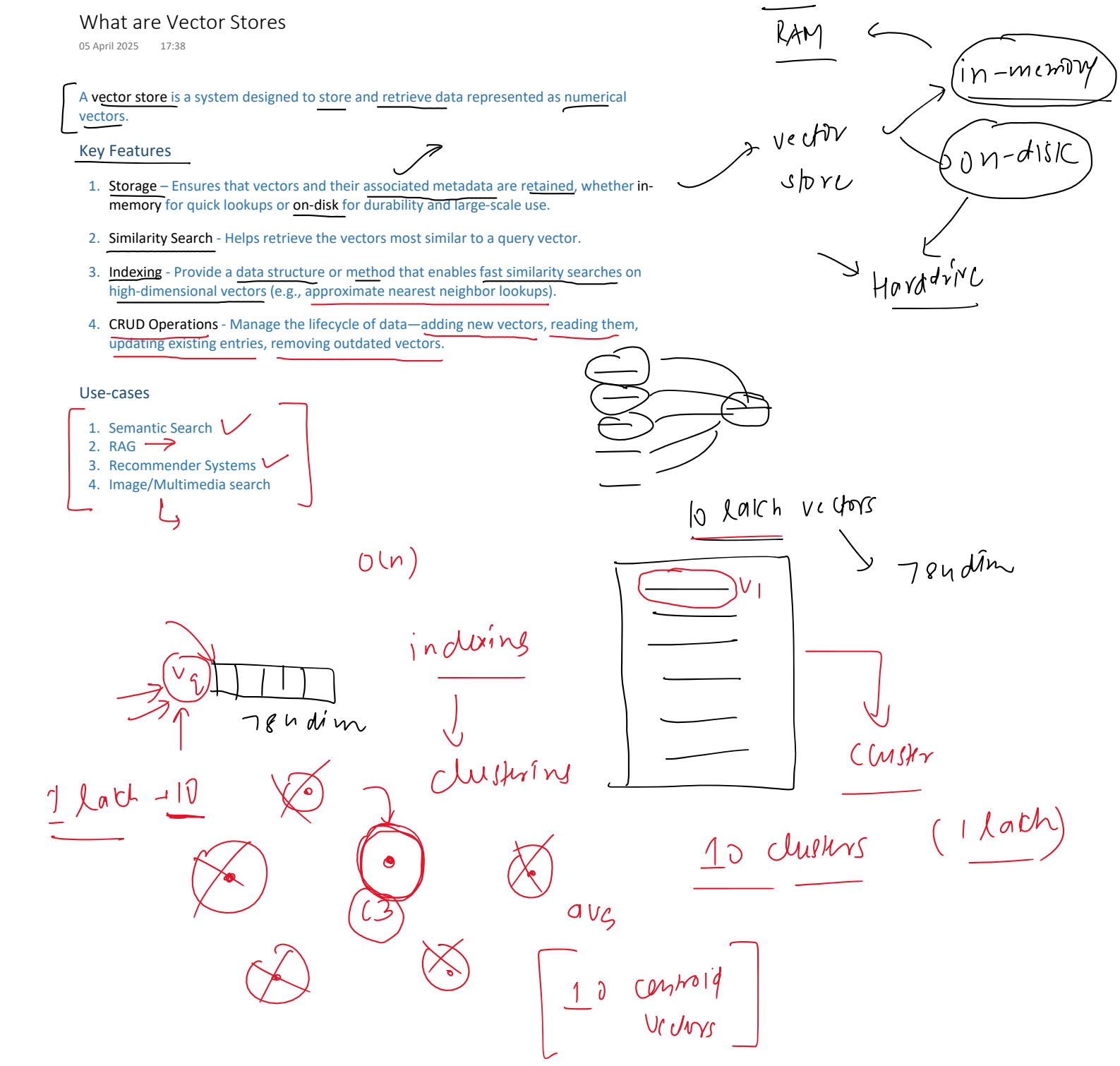
A vector store is a system designed to store and retrieve data represented as numerical vectors.

Key Features

1. Storage – Ensures that vectors and their associated metadata are retained, whether in-memory for quick lookups or on-disk for durability and large-scale use.
2. Similarity Search - Helps retrieve the vectors most similar to a query vector.
3. Indexing - Provide a data structure or method that enables fast similarity searches on high-dimensional vectors (e.g., approximate nearest neighbor lookups).
4. CRUD Operations - Manage the lifecycle of data—adding new vectors, reading them, updating existing entries, removing outdated vectors.

Use-cases

1. Semantic Search
2. RAG
3. Recommender Systems
4. Image/Multimedia search

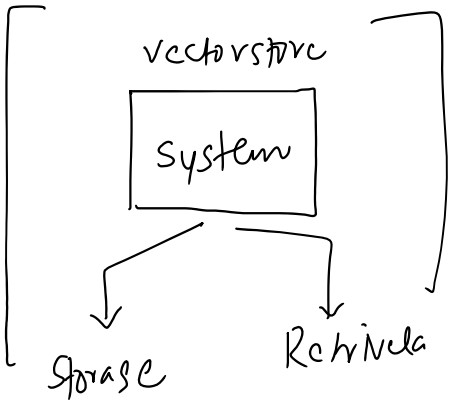


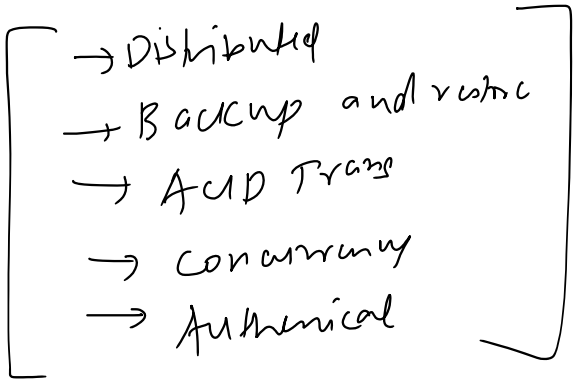
Vector Store Vs Vector Database

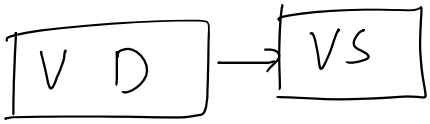
05 April 2025 17:40

- Vector Store
  - Typically refers to a lightweight library or service that focuses on storing vectors (embeddings) and performing similarity search.
  - May not include many traditional database features like transactions, rich query languages, or role-based access control.
  - Ideal for prototyping, smaller-scale applications
  - Examples: FAISS (where you store vectors and can query them by similarity, but you handle persistence and scaling separately).
- Vector Database
  - A full-fledged database system designed to store and query vectors.
  - Offers additional "database-like" features:
    - Distributed architecture for horizontal scaling
    - Durability and persistence (replication, backup/restore)
    - Metadata handling (schemas, filters)
    - Potential for ACID or near-ACID guarantees
    - Authentication/authorization and more advanced security
  - Geared for production environments with significant scaling, large datasets
  - Examples: Milvus, Qdrant, Weaviate. Pinecone

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



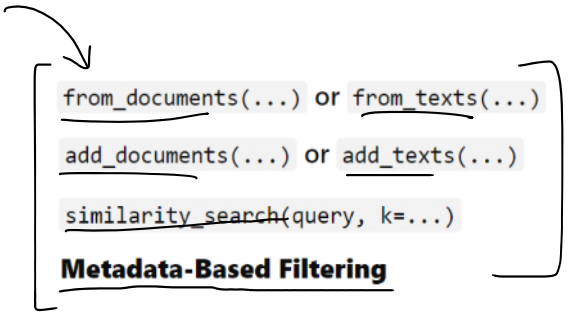


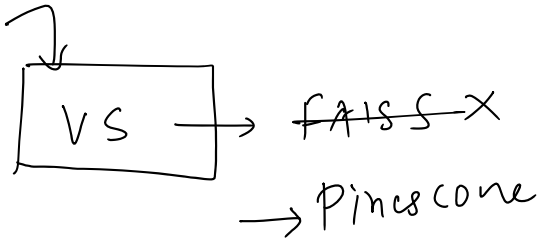


Vector Stores in LangChain

05 April 2025 17:41

- Supported Stores: LangChain integrates with multiple vector stores (FAISS, Pinecone, Chroma, Qdrant, Weaviate, etc.), giving you flexibility in scale, features, and deployment.
- Common Interface: A uniform Vector Store API lets you swap out one backend (e.g., FAISS) for another (e.g., Pinecone) with minimal code changes.
- Metadata Handling: Most vector stores in LangChain allow you to attach metadata (e.g., timestamps, authors) to each document, enabling filter-based retrieval.

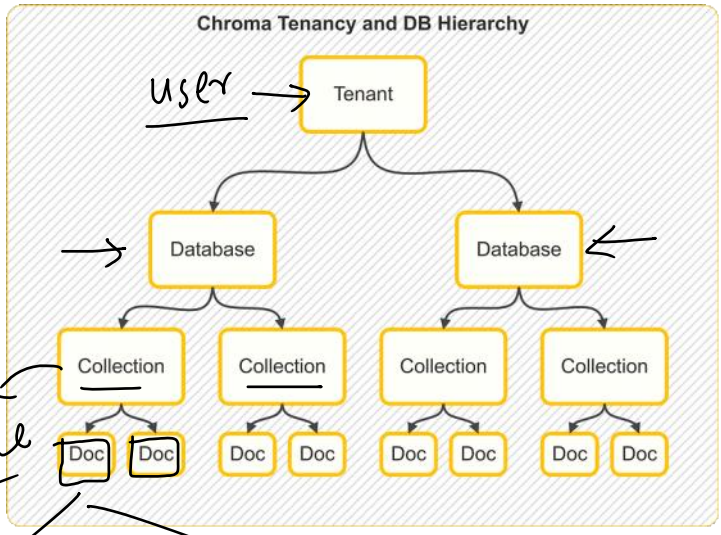




Chroma Vector Store

05 April 2025 17:41

Chroma is a lightweight, open-source vector database that is especially friendly for local development and small- to medium-scale production needs.

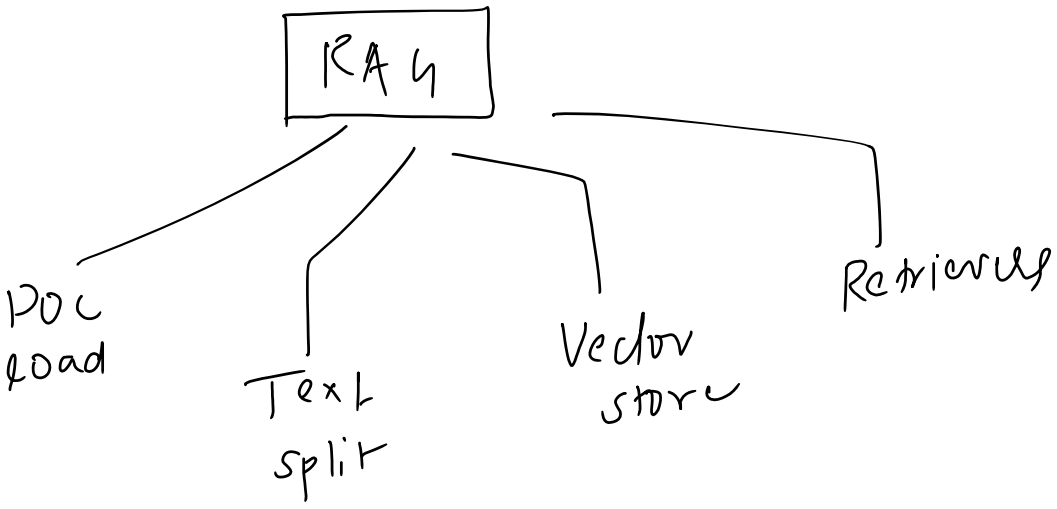


Code

05 April 2025 17:41

Recap

12 April 2025 13:51



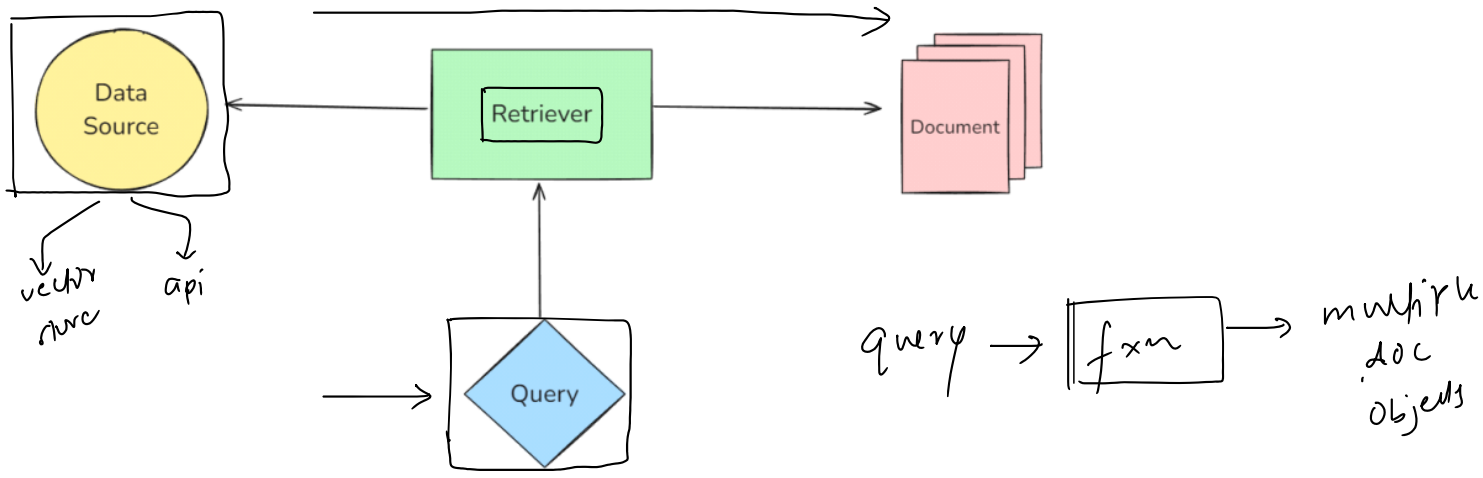
What are Retrievers

10 April 2025 07:56

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

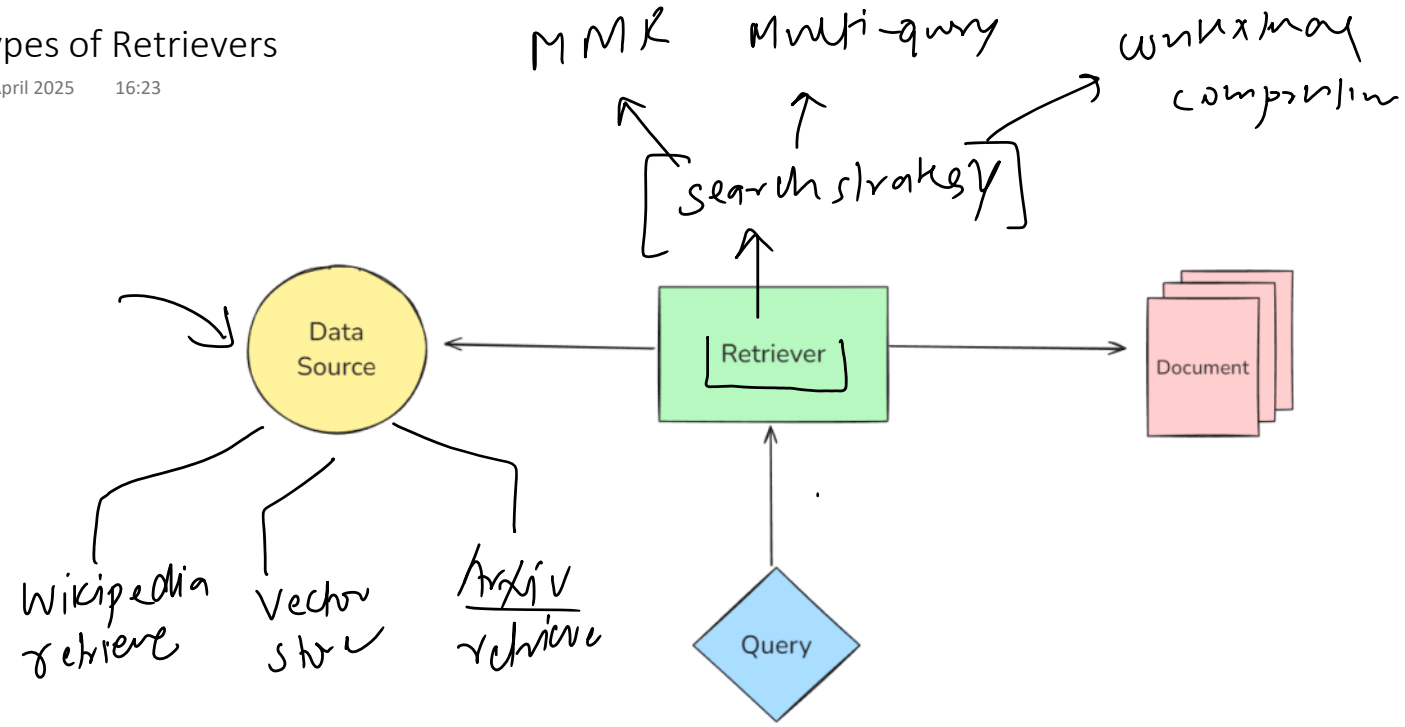
→ There are multiple types of retrievers

→ All retrievers in LangChain are runnables



Types of Retrievers

10 April 2025 16:23



Wikipedia Retriever → doc loader

10 April 2025 16:23

A Wikipedia Retriever is a retriever that queries the Wikipedia API to fetch relevant content for a given query.

How It Works

1. You give it a query (e.g., "Albert Einstein")
2. It sends the query to Wikipedia's API
3. It retrieves the **most relevant articles**
4. It returns them as LangChain `Document` objects

Vector Store Retriever

10 April 2025 16:24

A Vector Store Retriever in LangChain is the most common type of retriever that lets you search and fetch documents from a vector store based on semantic similarity using vector embeddings.

⚙️ How It Works

1. You store your documents in a **vector store** (like FAISS, Chroma, Weaviate)
2. Each document is converted into a **dense vector** using an **embedding model**
3. When the user enters a query:
    - It's also turned into a vector
    - The retriever compares the query vector with the stored vectors
    - It retrieves the top-k most similar ones

Maximal Marginal Relevance (MMR)

10 April 2025 16:24

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

MMR is an information retrieval algorithm 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 documents that are:

- All very similar to each other
- Repeating the same info
- Lacking diverse perspectives

MMR Retriever avoids that by:

- Picking the **most relevant document** first
- Then picking the next most relevant **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

Doc ID	Content
D1	"Climate change is causing glaciers to melt rapidly in the Arctic region."
D2	"Glaciers in the Arctic are melting at an alarming rate due to rising temperatures."
D3	"Deforestation in the Amazon is accelerating global climate change."
D4	"Climate change is increasing the frequency of wildfires in California."
D5	"Rising sea levels due to climate change threaten coastal cities like Mumbai and New York."

```
Top 3 results:
1. D1: Arctic glaciers melting
2. D2: Arctic glaciers melting
3. D3: Deforestation in Amazon
```

-diverse perspective

```
Top 3 results:
1. D1: Arctic glaciers melting
2. D4: Wildfires in California
3. D5: Rising sea levels in coastal cities
```

Multi-Query Retriever

10 April 2025 16:26

> **Sometimes a single query might not capture all the ways information is phrased in your documents.**

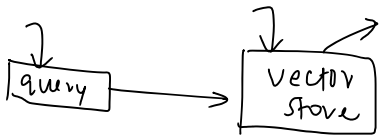
For example:

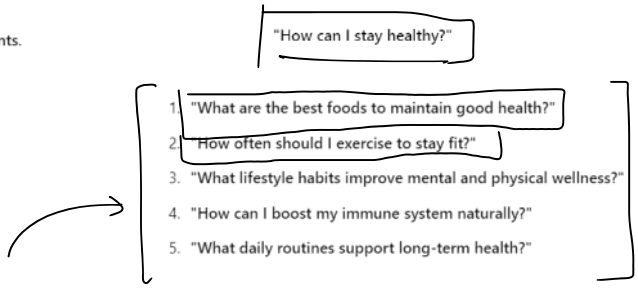
Query:

"How can I stay healthy?"

Could mean:

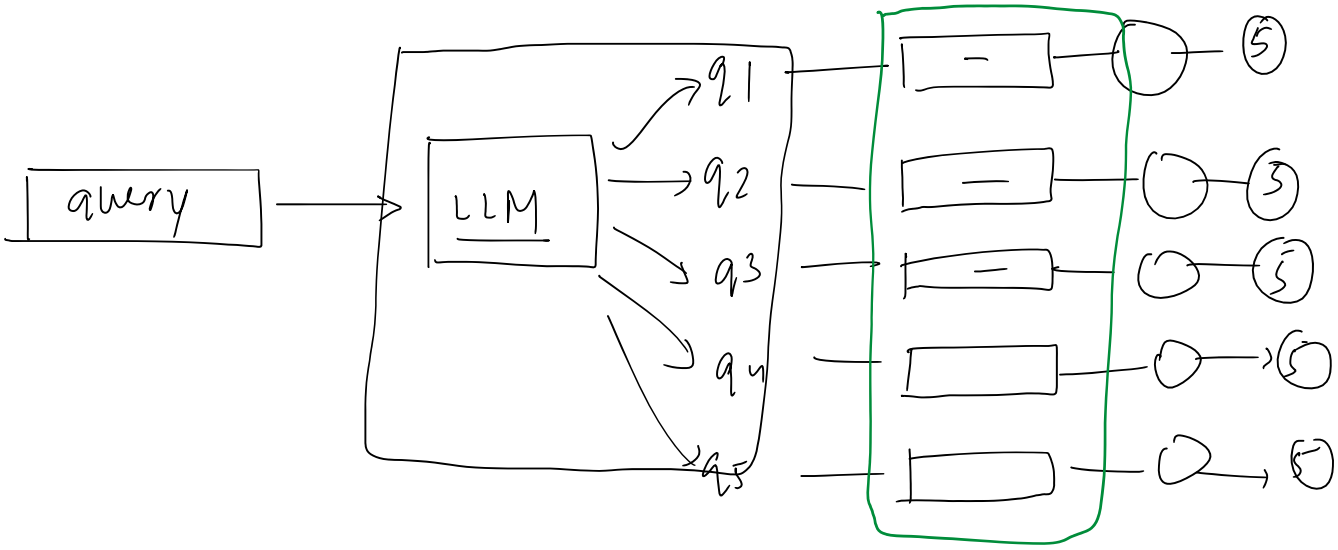
- What should I eat?
- How often should I exercise?
- How can I manage stress?





A simple similarity search might **miss documents** that talk about those things but don't use the word "healthy."

1. **Takes your original query**
2. **Uses an LLM (e.g., GPT-3.5)** to generate multiple semantically different versions of that query
3. **Performs retrieval for each sub-query**
4. **Combines and deduplicates the results**



Contextual Compression Retriever

10 April 2025 16:29

The Contextual Compression Retriever in LangChain is an advanced retriever that improves retrieval quality by compressing documents after retrieval — keeping only the relevant content based on the user's query.

❓ **Query:**

"What is photosynthesis?"

Retrieved Document (by a traditional retriever):

"The Grand Canyon is a famous natural site.
Photosynthesis is how plants convert light into energy.
Many tourists visit every year."

❌ **Problem:**

- The retriever returns the **entire paragraph**
- Only **one sentence** is actually relevant to the query
- The rest is **irrelevant noise** that wastes context window and may confuse the LLM

✅ **What Contextual Compression Retriever does:**

Returns only the relevant part, e.g.

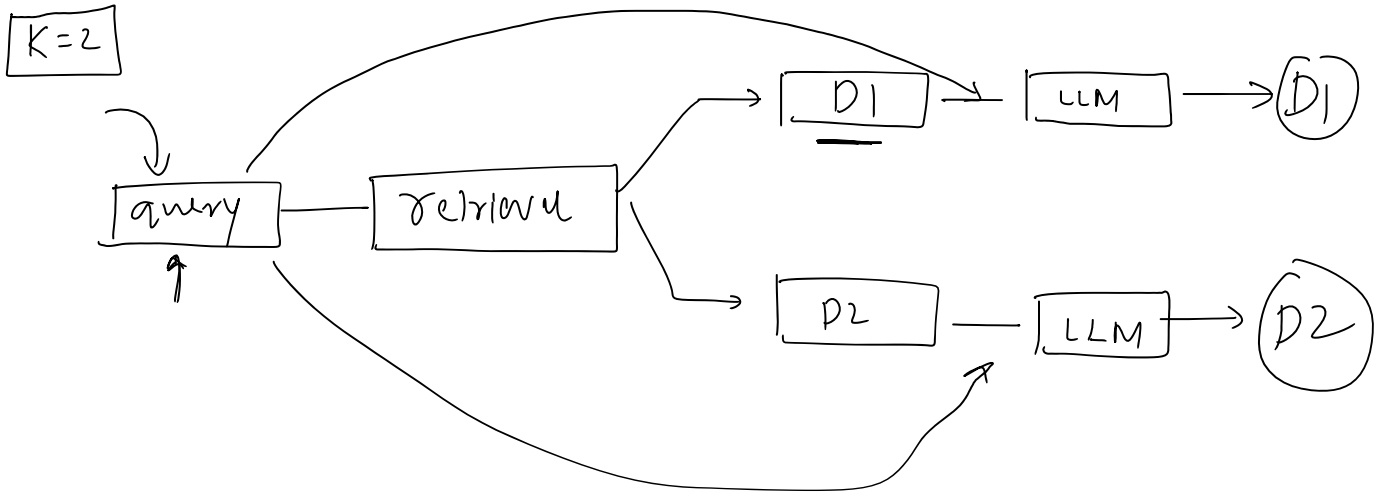
```
"Photosynthesis is how plants convert light into energy."
```

⚙️ **How It Works**

1. **Base Retriever** (e.g., FAISS, Chroma) retrieves N documents.
2. A **compressor** (usually an LLM) is applied to each document.
3. The compressor keeps **only the parts relevant to the query**.
4. Irrelevant content is discarded.

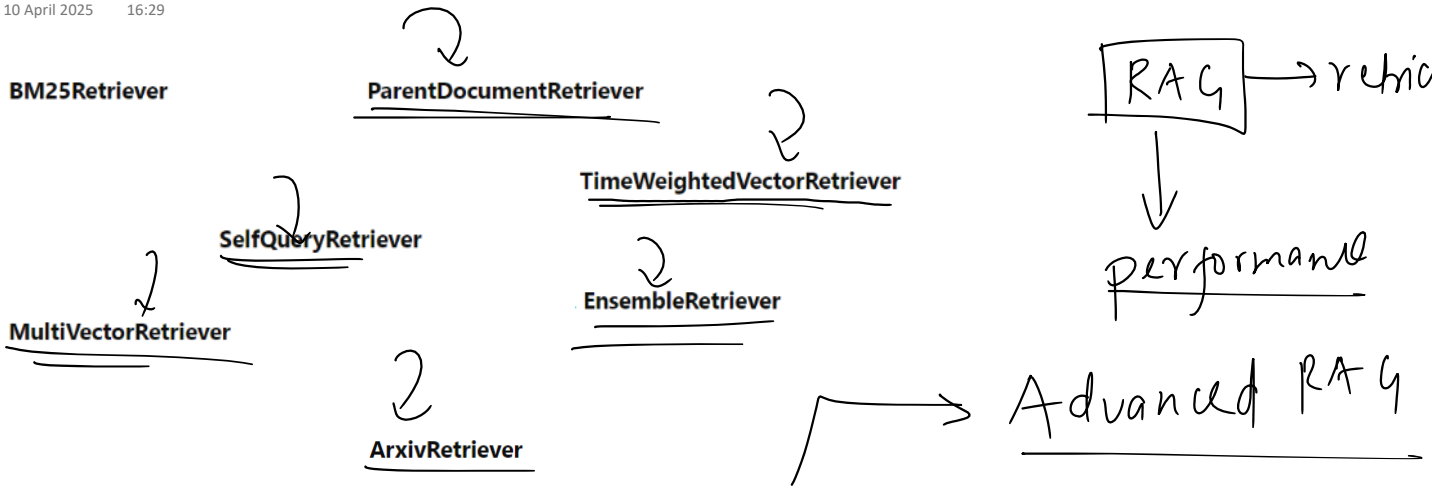
✅ **When to Use**

- Your documents are **long and contain mixed information**
- You want to **reduce context length** for LLMs
- You need to **improve answer accuracy** in RAG pipelines



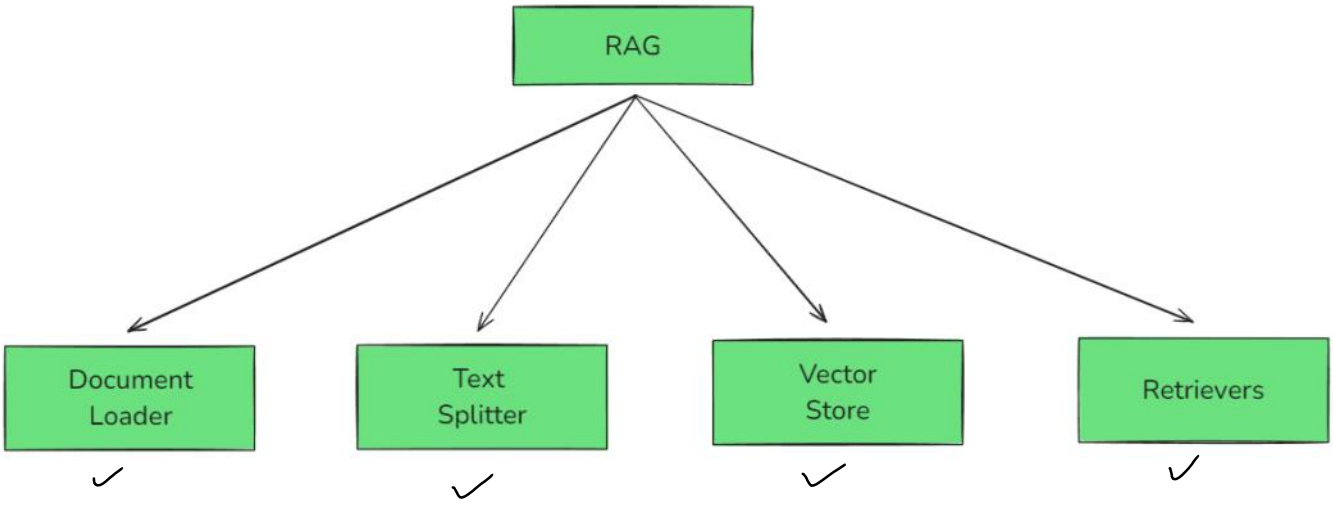
More Retrievers

10 April 2025 16:29



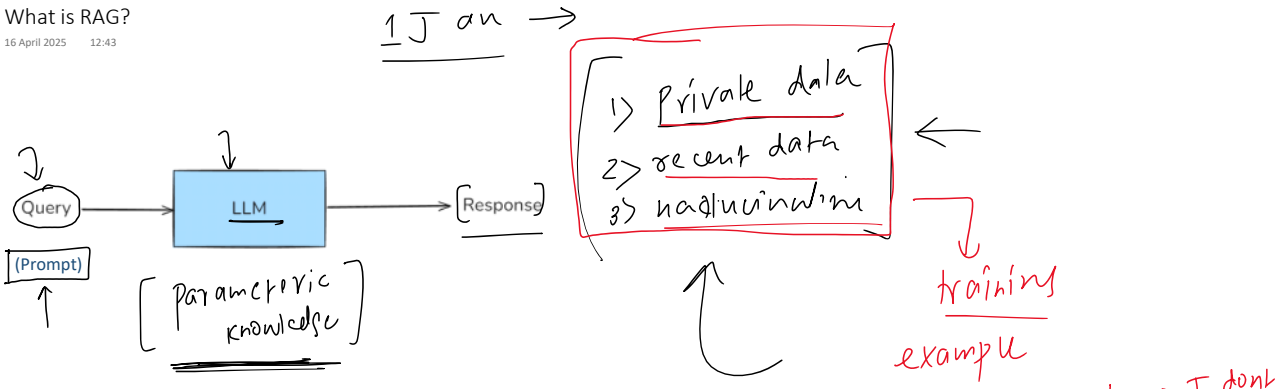
Recap

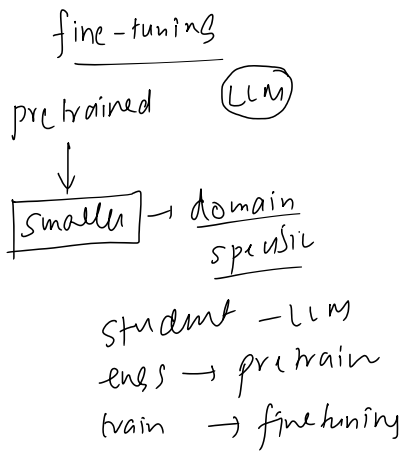
16 April 2025 12:45



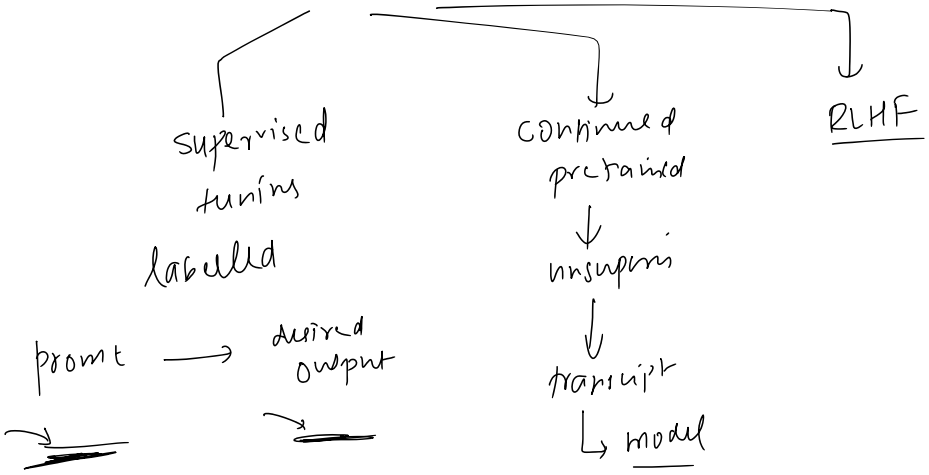
What is RAG?

16 April 2025 12:43





1. Collect data	A few hundred – few hundred-thousand carefully curated examples (prompts → desired outputs).
2. Choose a method	Full-parameter FT, LoRA/QLoRA, or parameter-efficient adapters.
3. Train for a few epochs	You keep the base weights frozen or partially frozen and update only a small subset (LoRA) **or** all weights (full FT).
4. Evaluate & safety-test	Measure exact-match, factuality, and hallucination rate against held-out data; red-team for safety.



1) LLM → train
2) technical expertise
3)

In context learning

In-Context Learning is a core capability of Large Language Models (LLMs) like GPT-3/4, Claude, and Llama, where the model learns to solve a task purely by seeing examples in the prompt—without updating its weights.

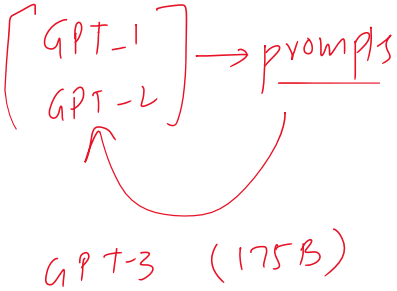
LLM → emergent property

```
Below are examples of texts labeled with their sentiment.
Use these examples to determine the sentiment of the final text.

Text: I love this phone. It's so smooth. → Positive
Text: This app crashes a lot. → Negative
Text: The camera is amazing! → Positive
```

LLM → emergent property

```
Use these examples to determine the sentiment of the final text.

Text: I love this phone. It's so smooth. → Positive
Text: This app crashes a lot. → Negative
Text: The camera is amazing! → Positive

Text: I hate the battery life. →
```

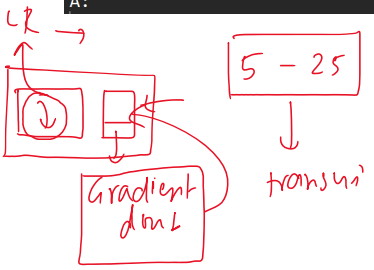
An **emergent property** is a behaviour or ability that suddenly appears in a system when it reaches a certain scale or complexity—even though it was not explicitly programmed or expected from the individual components.




```
Label the named entities in the sentences (Person, Location, Organization):

Sentence: Elon Musk founded SpaceX.
Entities: [Elon Musk → Person], [SpaceX → Organization]

Sentence: The Eiffel Tower is in Paris.
Entities: [Eiffel Tower → Location], [Paris → Location]

Sentence: Sundar Pichai leads Google.
Entities:
```

```
Solve the math problems step by step:

Q: John has 3 apples. He buys 4 more. How many apples does he have now?
A: John had 3 apples. He bought 4 more. 3 + 4 = 7. → 7

Q: Sarah has 10 pens. She gives away 3. How many does she have left?
A: Sarah had 10 pens. She gave away 3. 10 - 3 = 7. → 7

Q: A pizza is cut into 8 slices. Alex eats 3 slices. How many are left?
A:
```

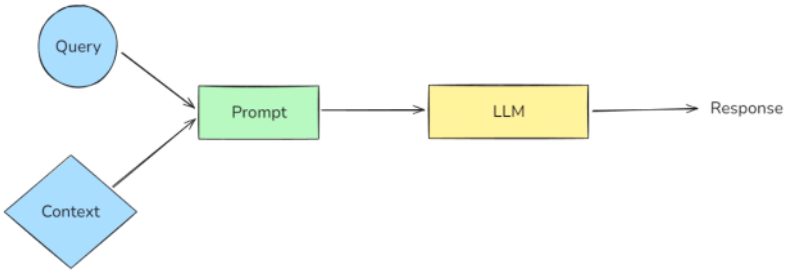
context



- Instead of just *example tasks*, retrieve **background information**, **facts**, **documents**, **product manuals**, etc.
- Inject that into the prompt to **augment the model's knowledge**

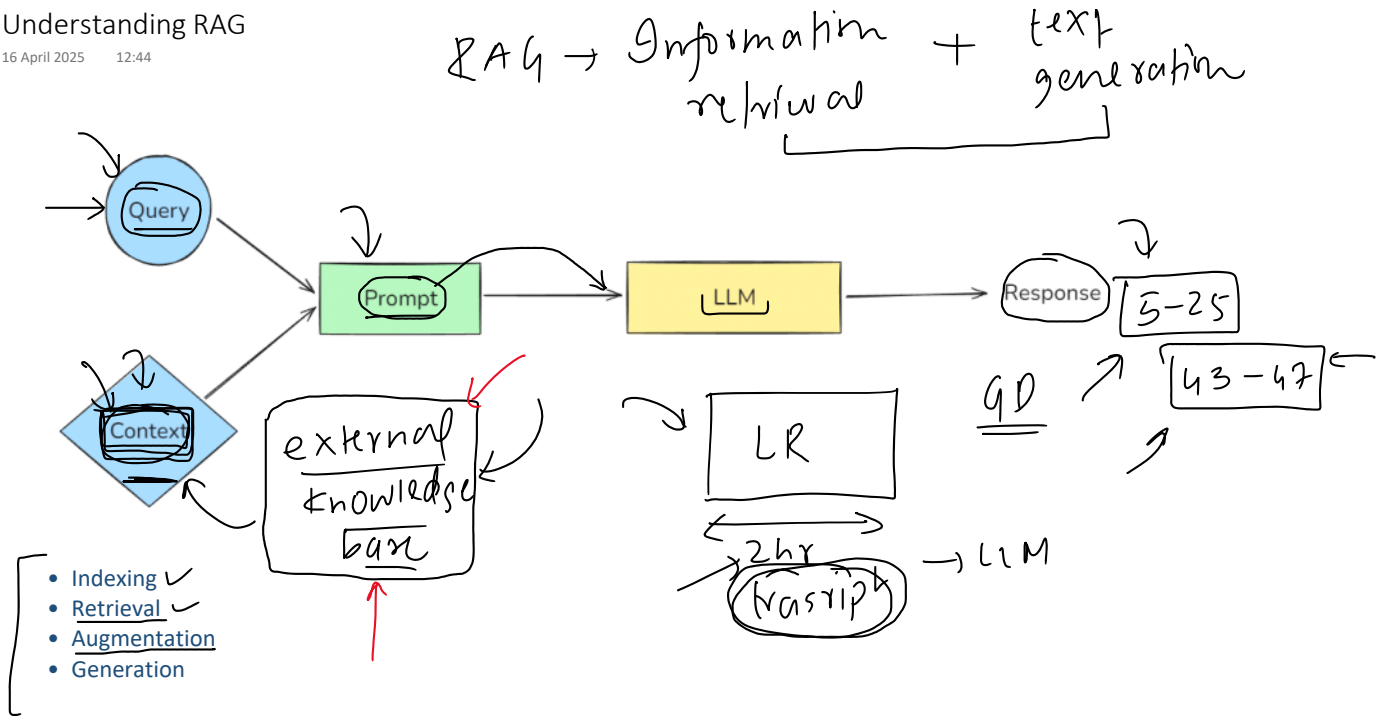
```
"""You are a helpful assistant.
Answer the question ONLY from the provided context.
If the context is insufficient, just say you don't know.

{context}
Question: {question}"""
```

RAG is a way to make a language model (like ChatGPT) smarter by giving it extra information at the time you ask your question.



Understanding RAG

16 April 2025 12:44



Indexing - **Indexing** is the process of **preparing your knowledge base so that it can be efficiently searched** at query time. This steps consists of 4 sub-steps.

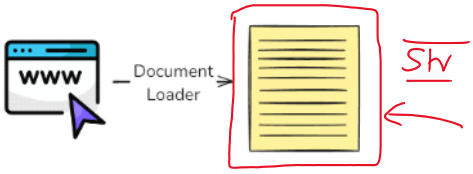
1. Document Ingestion - You load your source knowledge into memory.

   Examples:

   - PDF reports, Word documents
   - YouTube transcripts, blog pages
   - GitHub repos, internal wikis
   - SQL records, scraped webpages

   Tools:

   - LangChain loaders (`PyPDFLoader`, `YoutubeLoader`, `WebBaseLoader`, `GitLoader`, etc.)



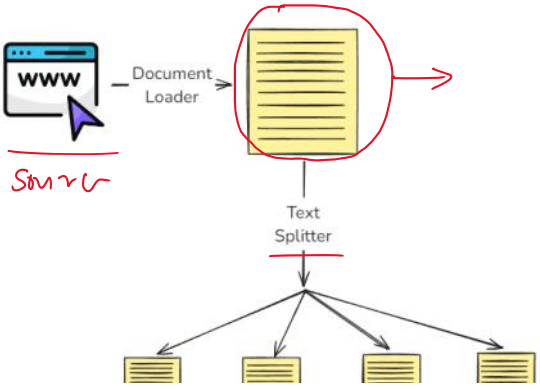
2. Text Chunking - Break large documents into small, semantically meaningful **chunks**

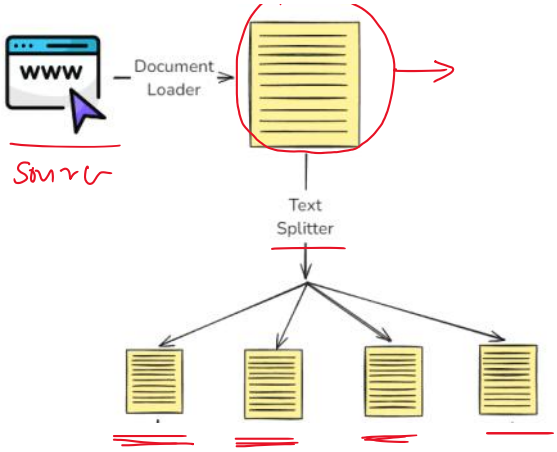
   Why chunk?

   - LLMs have context limits (e.g., 4K–32K tokens)
   - Smaller chunks are more focused → better semantic search

   Tools:

   - `RecursiveCharacterTextSplitter`, `MarkdownHeaderTextSplitter`, `SemanticChunker`





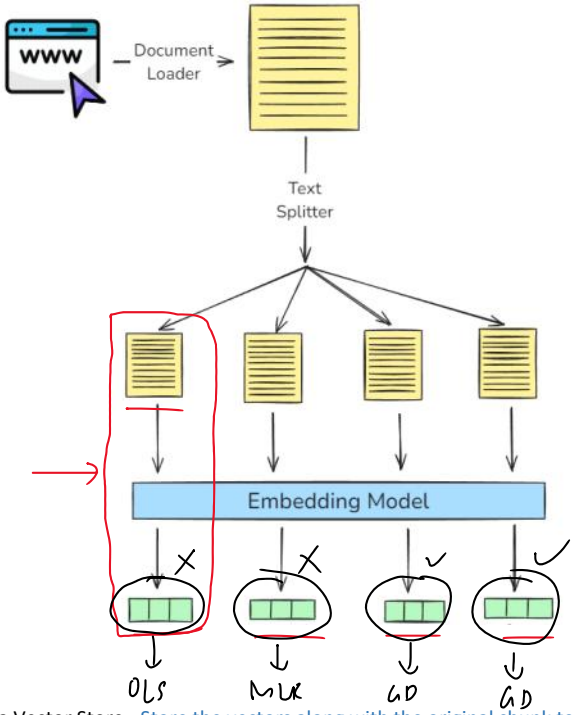
3. Embedding Generation - Convert each chunk into a **dense vector** (embedding) that captures its meaning.

   🔍 Why embeddings?

   - Similar ideas land close together in vector space
   - Allows fast, fuzzy semantic search

   🔧 Tools:

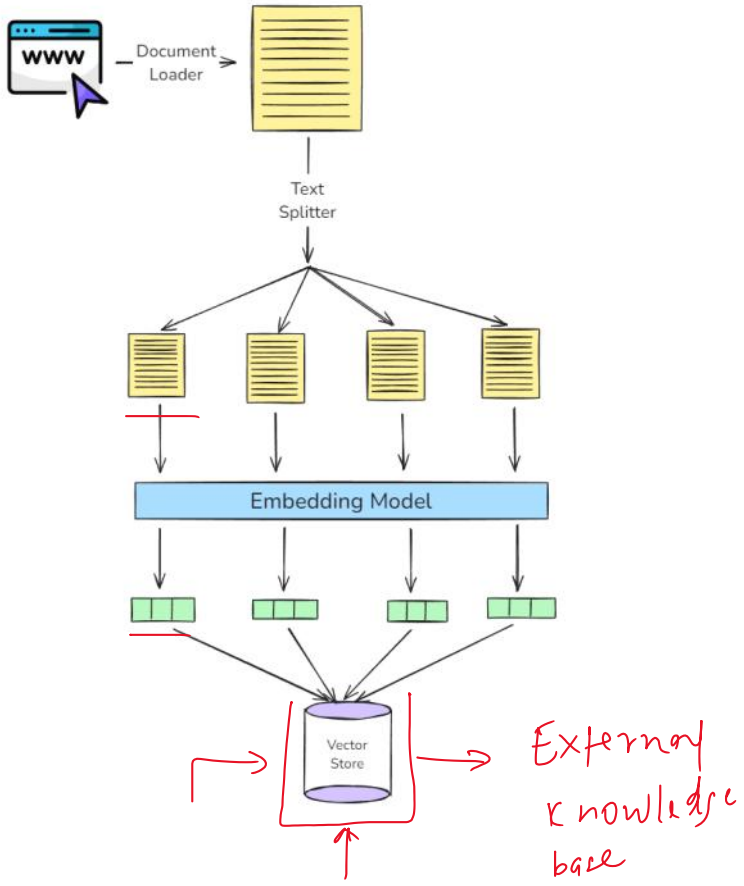
   - `OpenAIEmbeddings`, `SentenceTransformerEmbeddings`, `InstructorEmbeddings`, etc.



4. Storage in a Vector Store - Store the vectors along with the original chunk text + metadata in a **vector database**.

   🔧 Vector DB options:

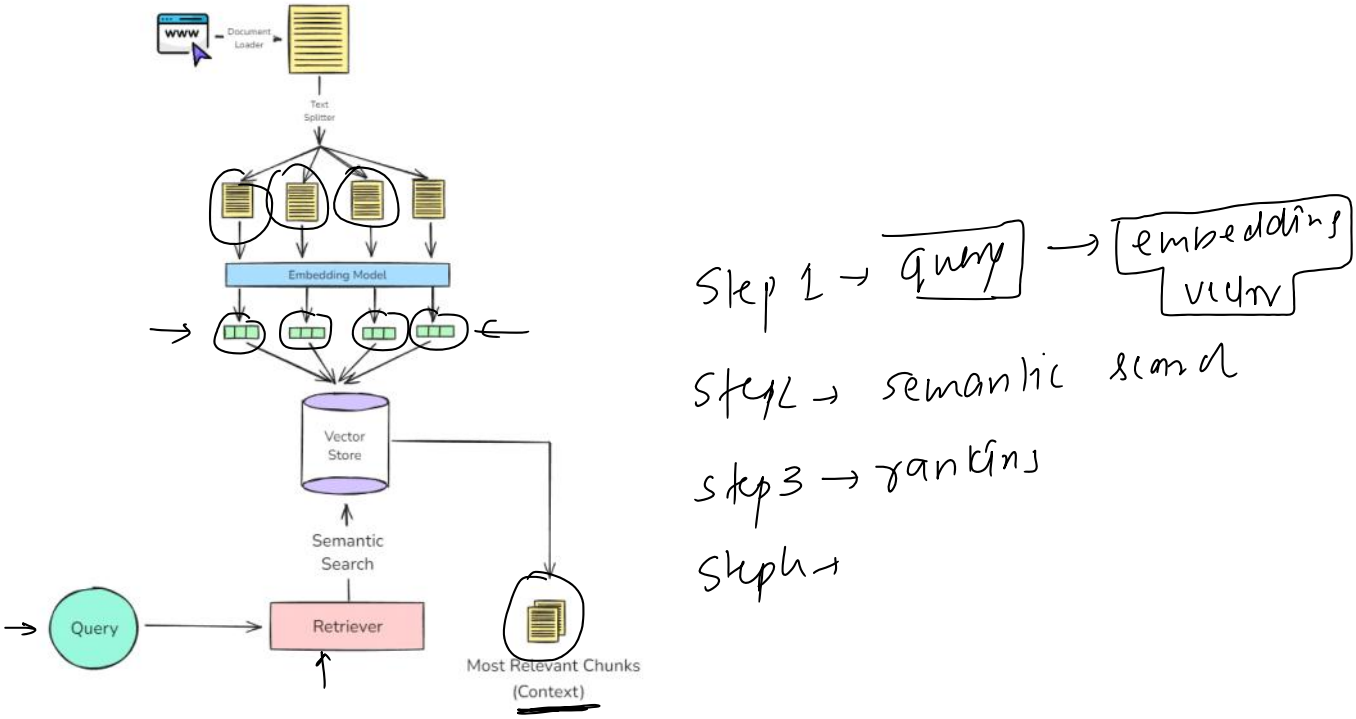
   - Local: `FAISS`, `Chroma`
   - Cloud: `Pinecone`, `Weaviate`, `Milvus`, `Qdrant`



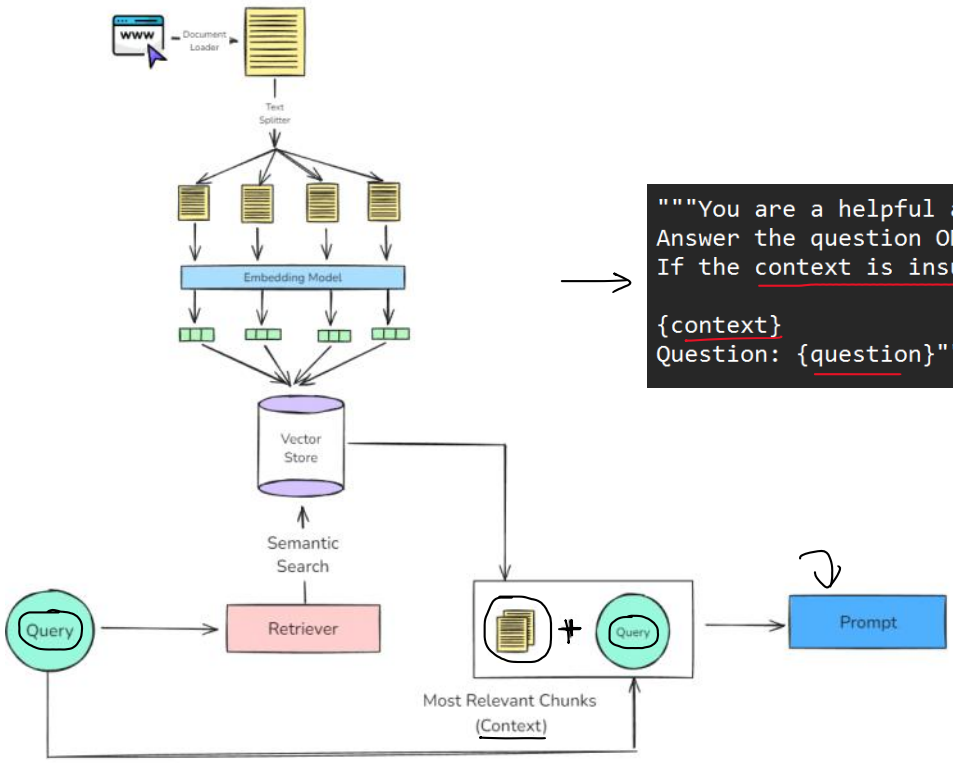
Retrieval - **Retrieval** is the *real-time* process of **finding the most relevant pieces of information** from a pre-built index (created during indexing) based on the user's question.

It's like asking:

> "From all the knowledge I have, which 3–5 chunks are most helpful to answer this query?"

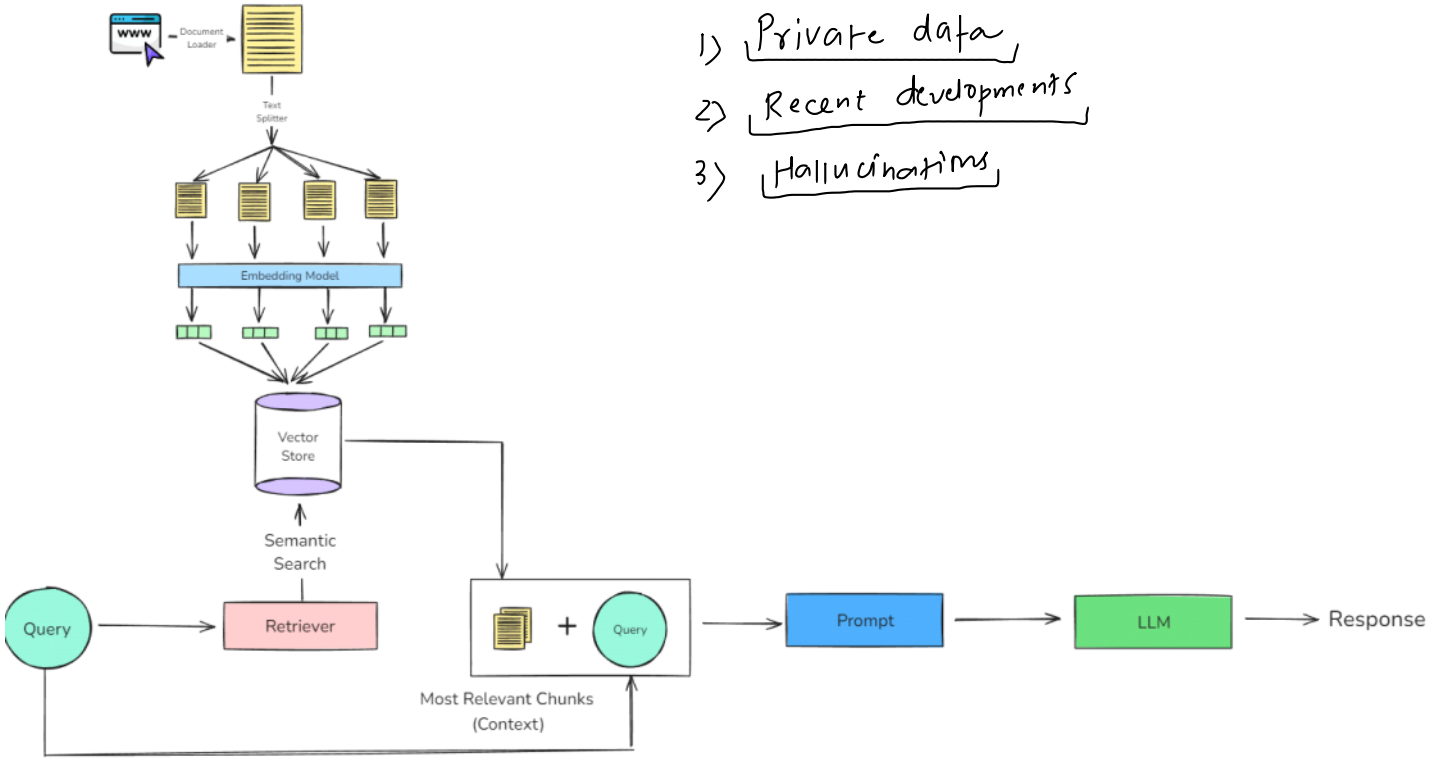


Augmentation - **Augmentation** refers to the step where the **retrieved documents** (chunks of relevant context) are **combined with the user's query** to form a new, enriched prompt for the LLM.




```
"""You are a helpful assistant.
Answer the question ONLY from the provided context.
If the context is insufficient, just say you don't know.

{context}
Question: {question}"""
```

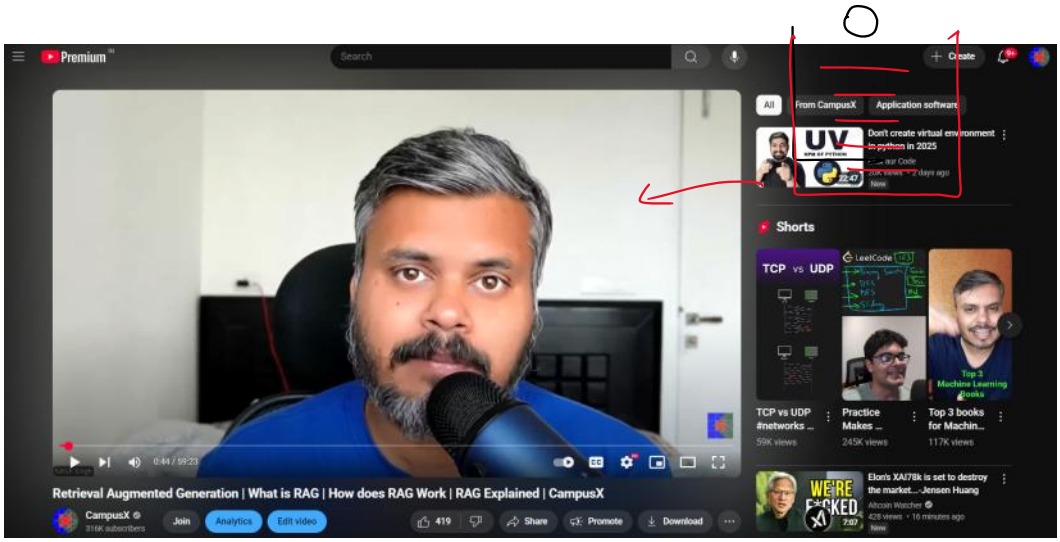
Generation - **Generation** is the final step where a **Large Language Model (LLM)** uses the **user's query** and the **retrieved & augmented context** to generate a response.



1) Private data
2) Recent developments
3) Hallucinations

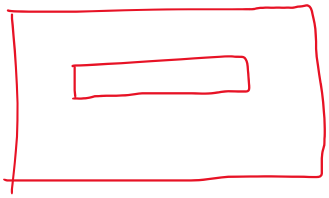
Problem Statement

21 April 2025 08:55



→ Chrome plugin

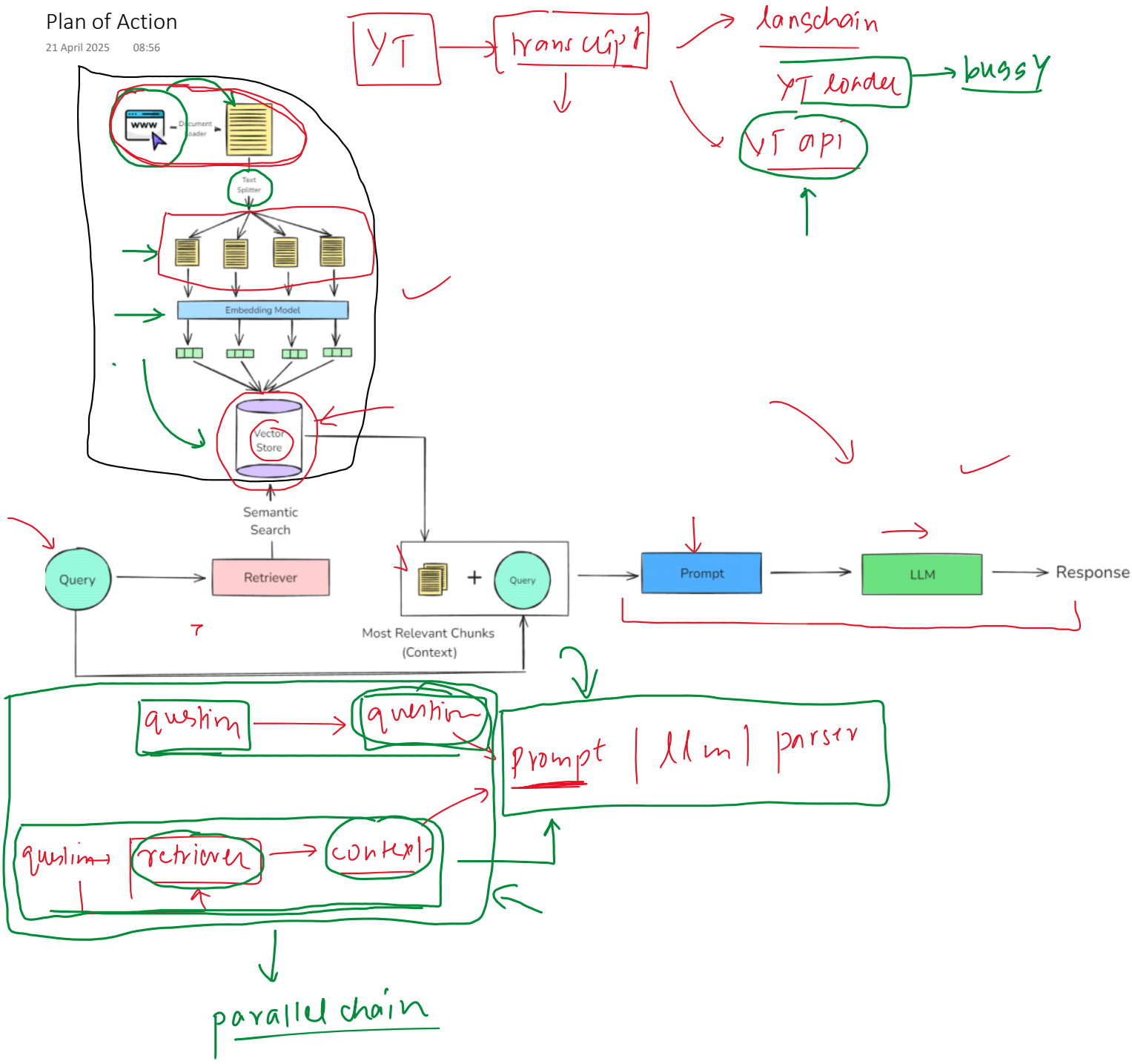
html/css/js



Streamlit →

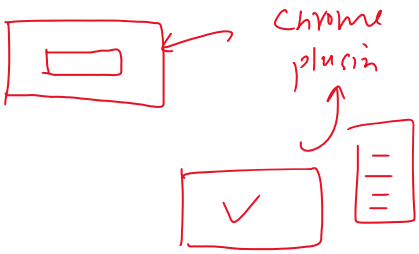
Plan of Action

21 April 2025 08:56



Improvements

22 April 2025 08:50

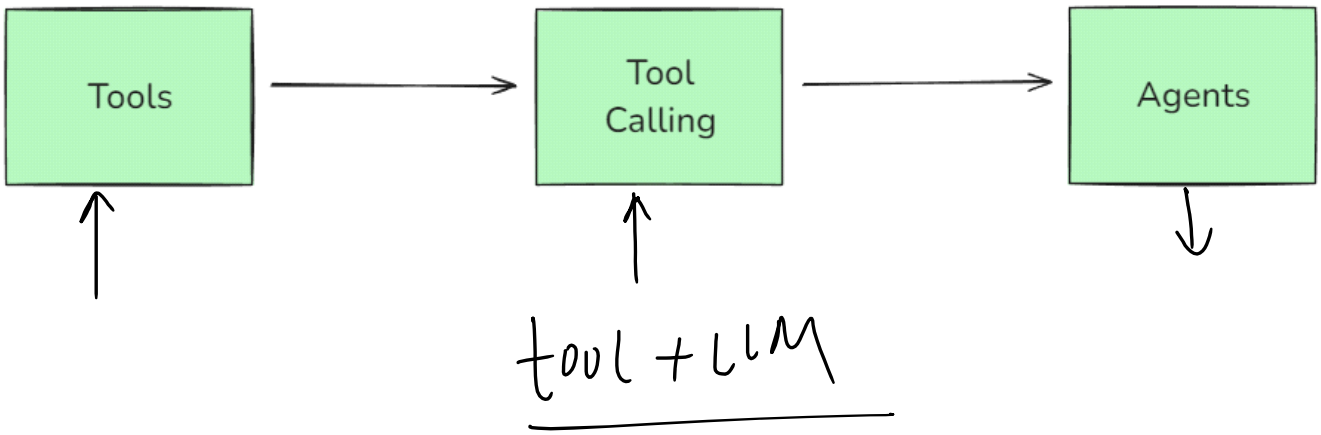


1. UI based enhancements
2. Evaluation
   a. Ragas
   b. LangSmith
3. Indexing
   a. Document Ingestion
   b. Text Splitting
   c. Vector Store — Pine Cone
4. Retrieval
   a. Pre-Retrieval
      i. Query rewriting using LLM
      ii. Multi-query generation
      iii. Domain aware routing
   b. During Retrieval
      i. MMR
      ii. Hybrid Retrieval
      iii. Reranking
   c. Post-Retrieval
      i. Contextual Compression
5. Augmentation
   a. Prompt Templating
   b. Answer grounding
   c. Context window optimization
6. Generation
   a. Answer with Citation
   b. Guard railing
7. System Design
   a. Multimodal
   b. Agentic
   c. Memory based

Metric	What It Measures
faithfulness	Is the answer grounded in the retrieved context?
answer_relevancy	Is the answer relevant to the user's question?
context_precision	How much of the retrieved context is actually useful?
context_recall	Did we retrieve all necessary information?

Overview

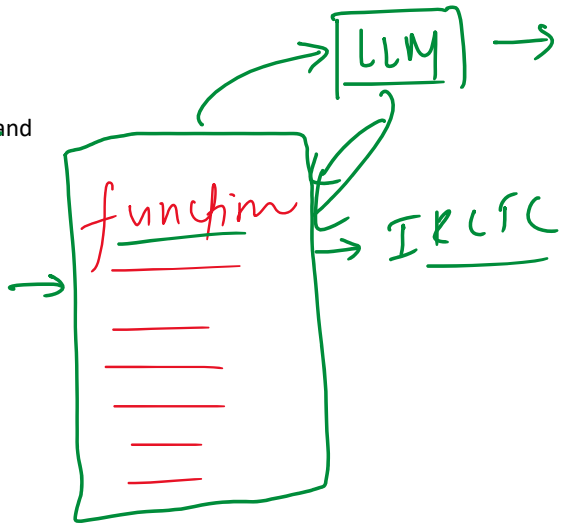
23 April 2025 15:51



What is a Tool?

23 April 2025 15:45

A tool is just a Python function (or API) that is packaged in a way the LLM can understand and call when needed.

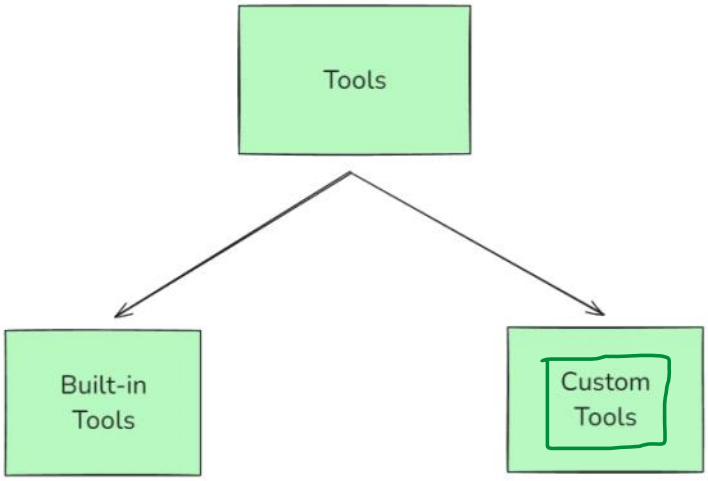


LLMs (like GPT) are great at:

- Reasoning (think)
- Language generation (speak)

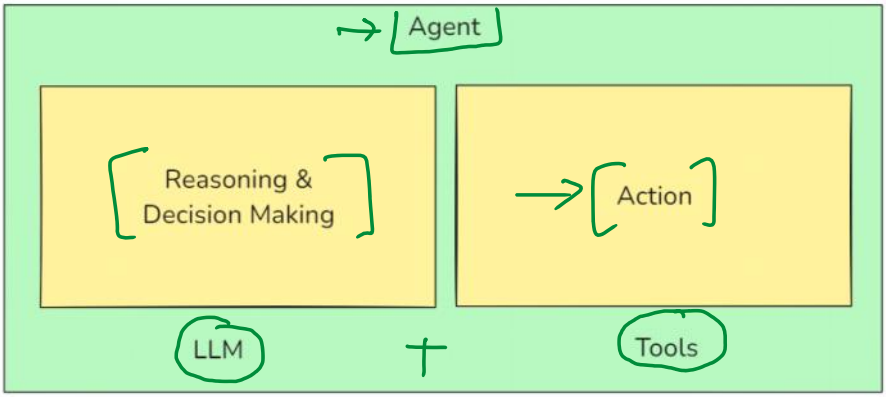
But they **can't do things like:**

- Access live data (weather, news)
- Do reliable math
- Call APIs
- Run code
- Interact with a database



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 APIs to achieve a goal.



Built-in Tools

23 April 2025 15:46

A built-in tool is a tool that LangChain already provides for you —it's pre-built, production-ready, and requires minimal or no setup.

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

`DuckDuckGoSearchRun`	Web search via DuckDuckGo
`WikipediaQueryRun`	Wikipedia summary
`PythonREPLTool`	Run raw Python code
`ShellTool`	Run shell commands
`RequestsGetTool`	Make HTTP GET requests
`GmailSendMessageTool`	Send emails via Gmail
`SlackSendMessageTool`	Post message to Slack
`SQLDatabaseQueryTool`	Run SQL queries

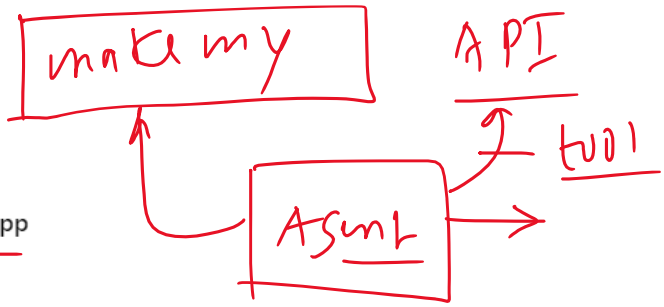
Custom Tools

23 April 2025 15:46

A custom tool is a tool that you define yourself.

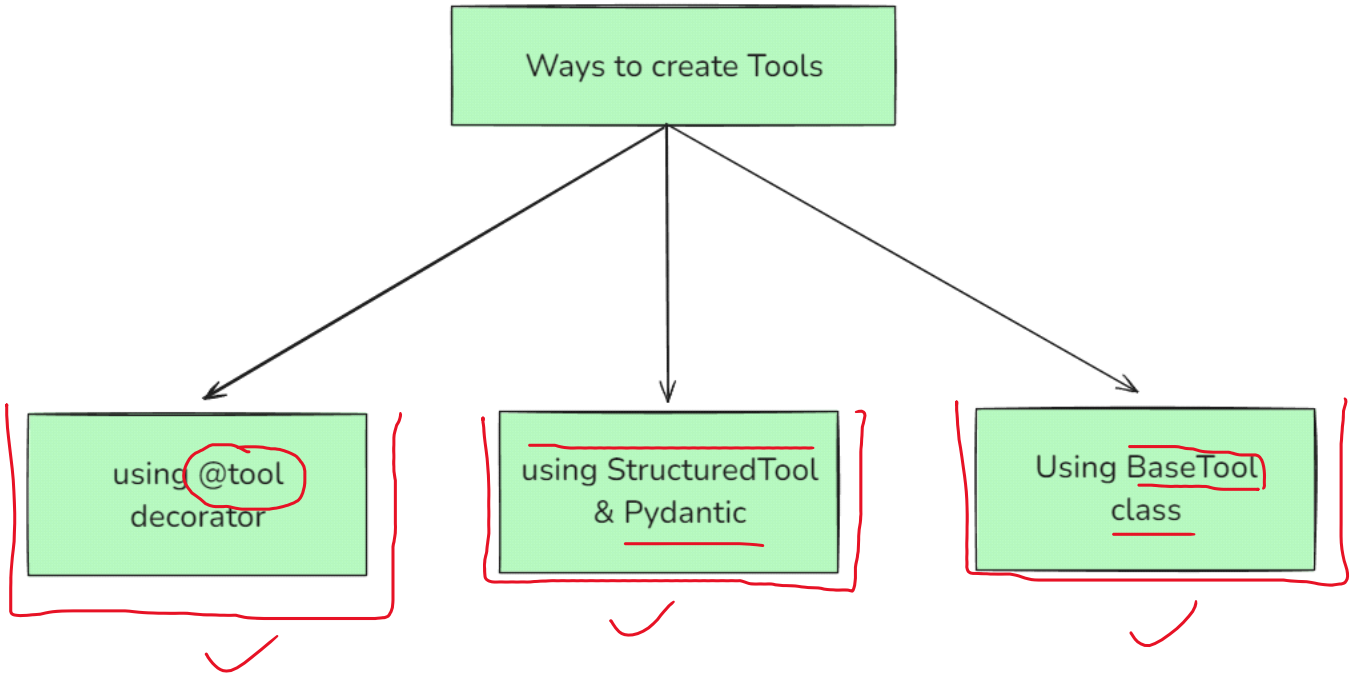
Use them when:

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



Ways to create Custom Tools

23 April 2025 15:46



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 **@tool, StructuredTool** are built on top of BaseTool

Toolkits

23 April 2025 15:47

A toolkit is just a collection (bundle) of related tools that serve a common purpose — packaged together for convenience and **reusability**.

In LangChain:

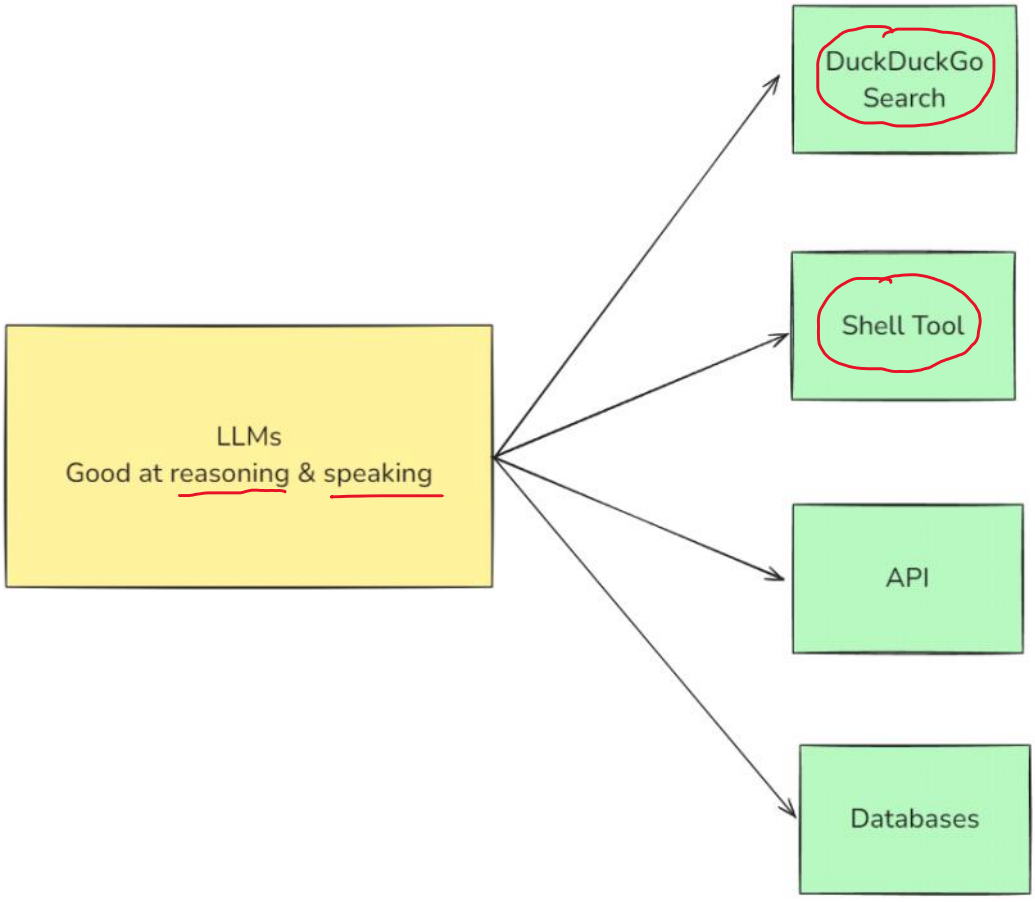
- A toolkit might be: GoogleDriveToolKit
- And it can contain the following tools

- `GoogleDriveCreateFileTool` : Upload a file
- `GoogleDriveSearchTool` : Search for a file by name/content
- `GoogleDriveReadFileTool` : Read contents of a file

Quick Revision

25 April 2025 16:18

TOOLS

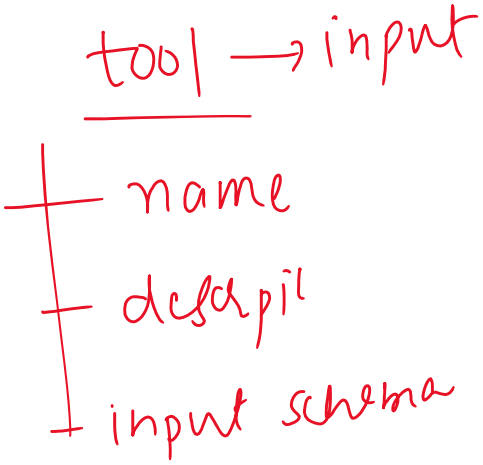


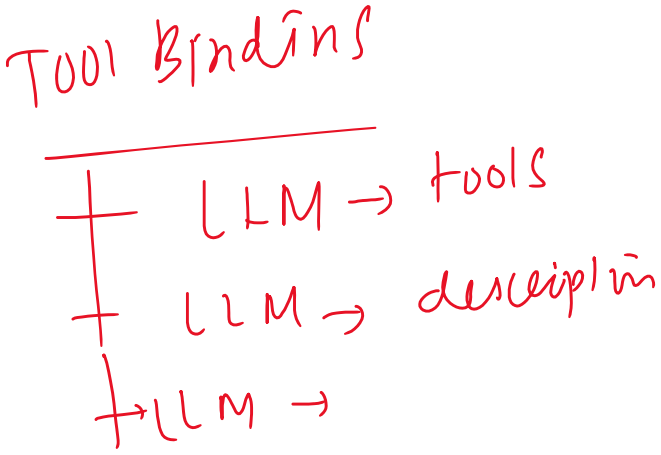
Tool Binding

25 April 2025 16:45

Tool Binding is the step where you **register tools with a Language Model (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

25 April 2025 16:54

Tool Calling is the process where the **LLM (language model)** 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**
- and the **arguments** to call it with

🚨 The 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**.

> "What's 8 multiplied by 7?"

The LLM responds with a **tool call**:

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

Tool creation
Tool bindings

name : multiply
schema

Tool Execution

25 April 2025 17:12

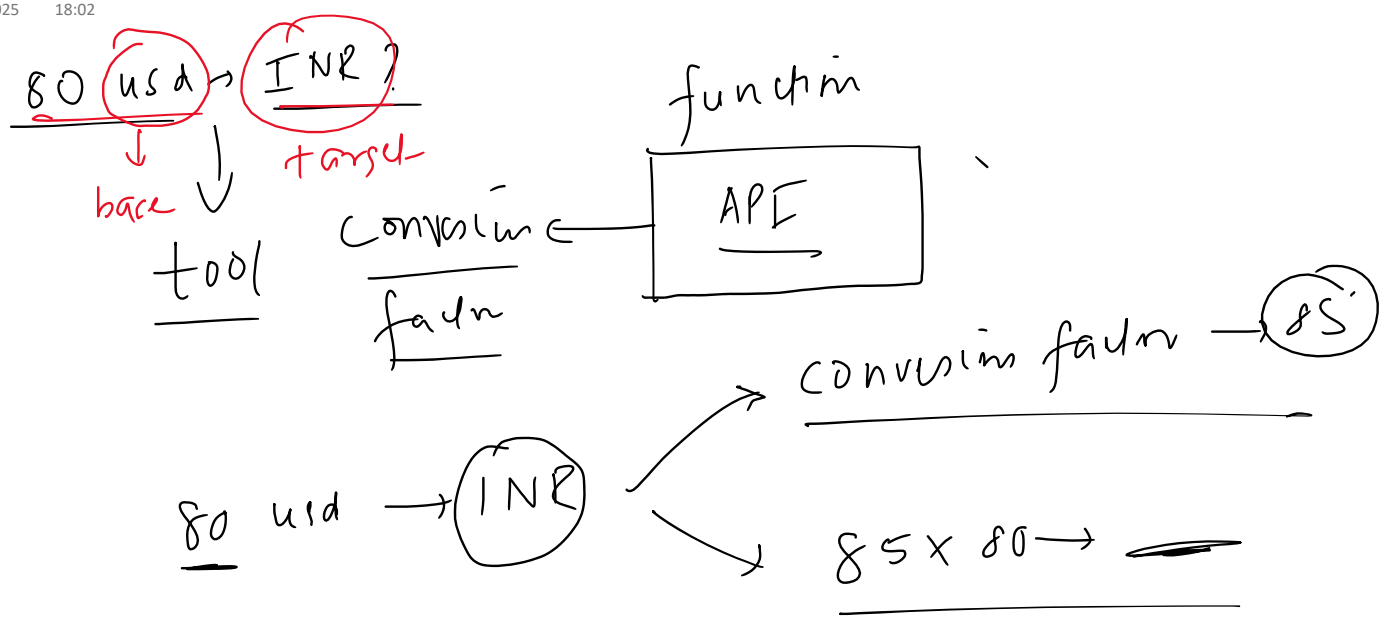
Tool Execution is the step where the **actual Python function (tool)** is run using the input arguments that the **LLM suggested during tool calling**.

In simpler words:

> 🧠 The LLM says:
> *"Hey, call the `multiply` tool with a=8 and b=7."*
>
> ⚙️ **Tool Execution** is when *you or LangChain actually run:*
> `multiply(a=8, b=7)`
> → and get the result: `56`

Currency Conversion Tool

25 April 2025 18:02



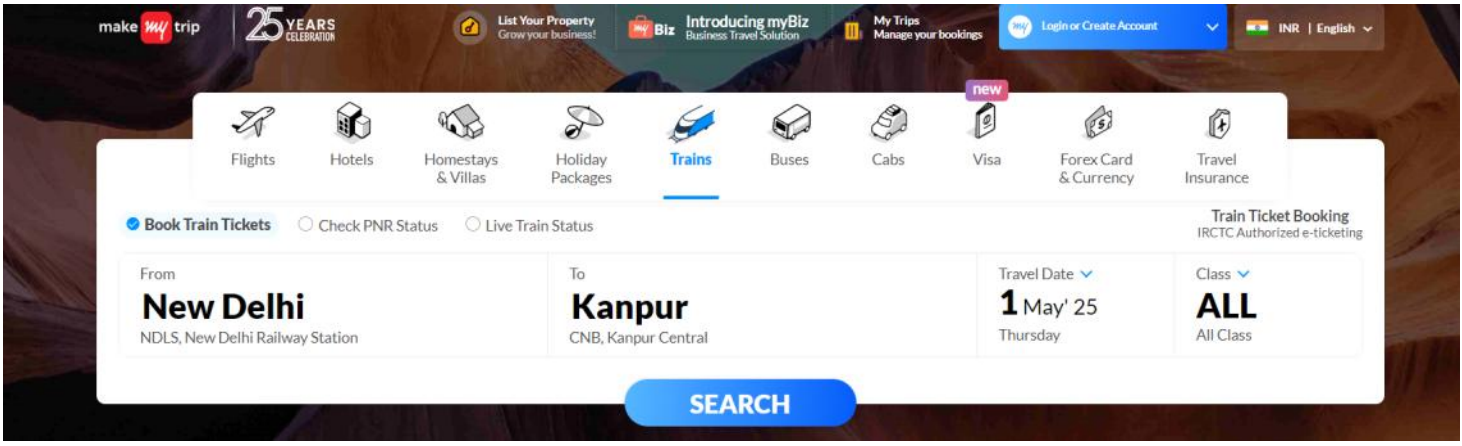
> ❌ "LLM, do **not** try to fill this argument." ✅ "I (the developer/runtime) will inject this value after running earlier tools."

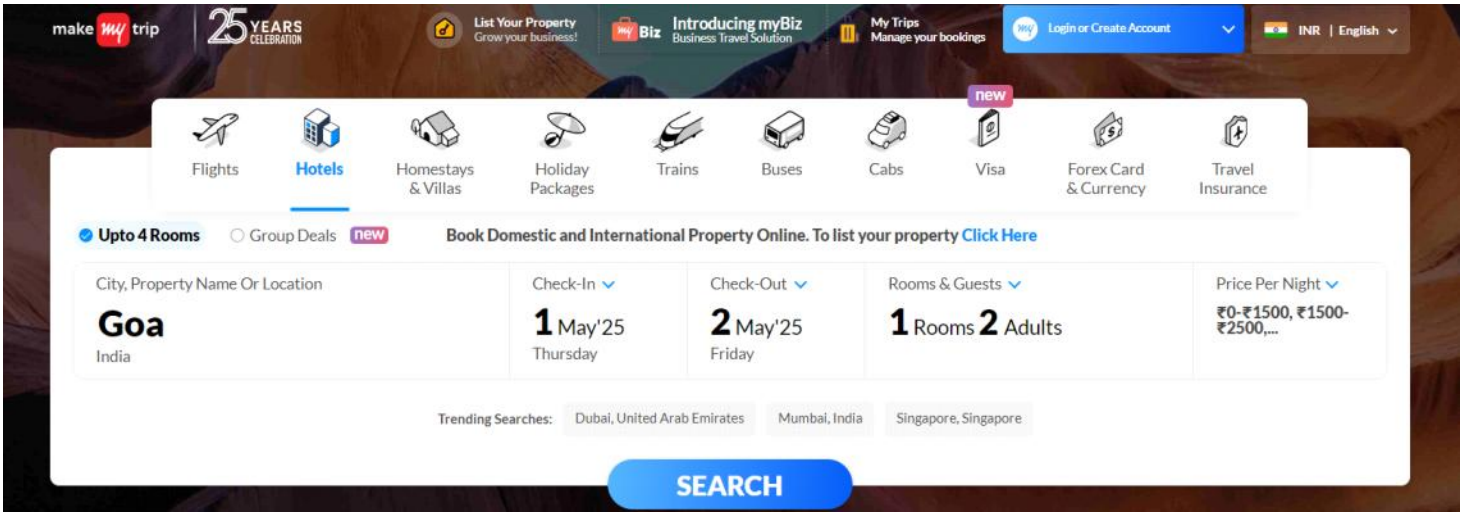
AI Agent → tools / tool calling (X)

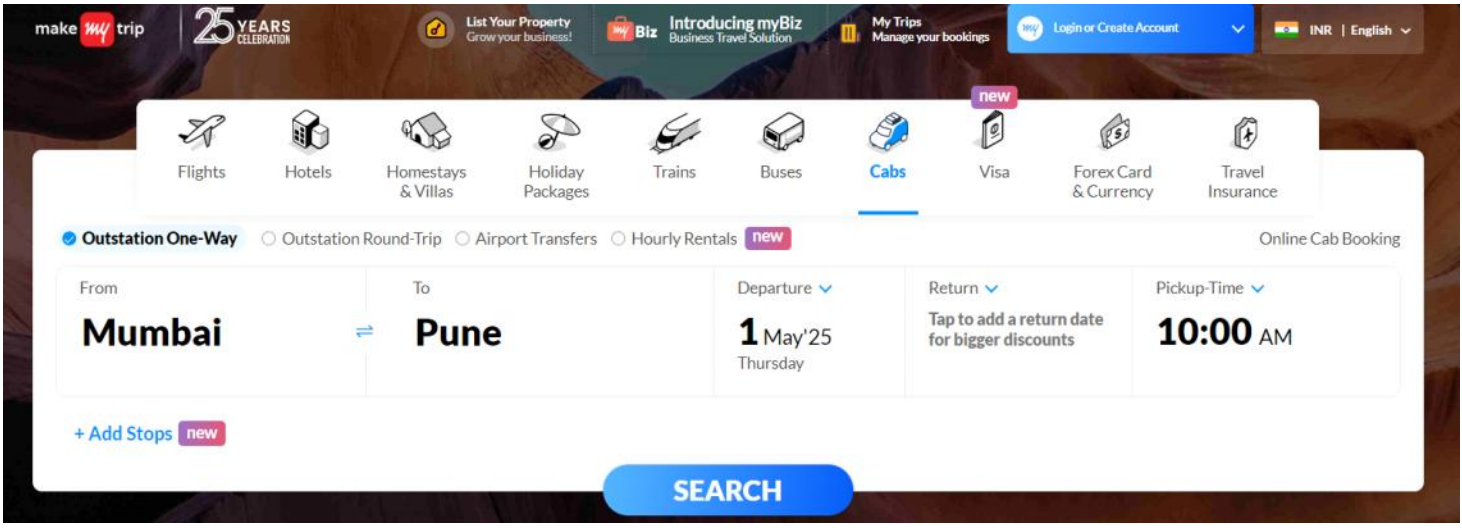
1. **User says**: "Convert 10 USD to INR."
2. **LLM thinks**: "I don't know the rate. First, let me call `get_conversion_factor`."
3. **Tool result comes**: 85.3415
4. **LLM looks at result, THINKS again**: "Now I know the rate, next I should call `convert` with 10 and 85.3415."
5. **Tool result comes**: 853.415 INR
6. **LLM summarizes**: "10 USD is 853.415 INR at current rate."

What are AI Agents

29 April 2025 22:53







✅ **User enters query:**

> "Can you create a budget travel itinerary from Delhi to Goa from 1st to 7th May?"

🧠 1. Understanding your intent

The agent internally interprets:

- ✅ Origin: Delhi
- ✅ Destination: Goa
- ✅ Dates: May 1–7
- ✅ Preference: Budget travel
- ✅ Goal: Plan complete itinerary + optimize cost

Agent thinks: *"I need to plan affordable travel, stay, local movement, and activities over 7 days."*

🔍 2. Research transport options

Agent (internally):

- Calls **IRCTC/train API** and **Flight APIs**
- Compares price, duration, and availability

Agent (to user):

> "The cheapest and most available option is the Goa Express from Delhi on April 30th night (sleeper class ₹800 or 3AC ₹1500).
> It arrives in Goa on May 1st evening. Would you like to go ahead?"

✅ User selects "3AC".

🏨 3. Search for affordable stay

Agent (internally):

- Uses hotel API or scraping tools
- Filters by:
  - Price < ₹700/night
  - Close to Baga or Anjuna beach
  - User reviews

Agent (to user):

> "I found a dorm room at The Hosteller, Baga for ₹650/night with good ratings. Shall I book this for 6 nights (May 1–7)?"

✅ User agrees.

🛵 4. **Plan local travel**

Agent (to user):

> "Scooter rentals are the most cost-effective in Goa.
> Should I pre-book a scooty for ₹300/day? (~₹1800 for 6 days)"

✅ User says yes.

🎯 5. **Generate activity plan**

Agent (internally):

- Calls sightseeing API / uses static local dataset
- Maps popular attractions across North & South Goa
- Builds a balanced plan:
  - Culture, relaxation, nightlife, shopping

Agent (to user):

> "Here's a draft plan:

- **May 2**: Baga, Calangute, Aguada Fort
- **May 3**: Old Goa churches, Fontainhas, local lunch
- **May 4**: South Goa day trip: Palolem, Cabo de Rama
- **May 5**: Anjuna flea market + sunset at Chapora
- **May 6**: Free day + beach cafe dinner"

✅ User loves it.

🛬 6. **Plan return**

Agent (to user):

> "Return options on May 7:

1. Train (₹800–₹1500, overnight)
2. Early morning flight (₹2800, 6:00 AM via SpiceJet)

Choose your preference."

✅ User picks train.

🧾 7. **Budget summary**

Agent (to user):

Item	Cost
Train (round trip)	₹3,000
Stay (6 nights)	₹3,900
Scooter + fuel	₹2,200
Food (local + budget)	₹2,500
Sightseeing & entry	₹1,000
Shopping/misc	₹1,500
Total	**~₹14,100**

> "Total trip cost: ₹14,100. Would you like to finalize the bookings?"

✅ User confirms.

📩 8. **Final Output**

Agent (to user):

> ✅ Bookings confirmed.
> 🧾 Receipt sent to your email.
> 🗓️ Calendar invite added for train departures.
> 📱 You'll get daily reminders and local recommendations during your stay.
> 💬 Need help anytime? Just ask!

> **An AI agent is an intelligent system that receives a high-level goal from a user, and autonomously plans, decides, and executes a sequence of actions by using external tools, APIs, or knowledge sources — all while maintaining context, reasoning over multiple steps, adapting to new information, and optimizing for the intended outcome.**

Goal-driven	You tell the agent *what you want*, not *how to do it*
Autonomous planning	Agent breaks down the problem and sequences tasks on its own
Tool-using	Agent calls APIs, calculators, search tools, etc.
Context-aware	Maintains memory across steps to inform future actions
Reasoning-capable	Makes decisions dynamically (e.g., "what to do next")
Adaptive	Rethinks plan when things change (e.g., API fails, no data)

Building an agent in LangChain

29 April 2025 22:53

Explanation

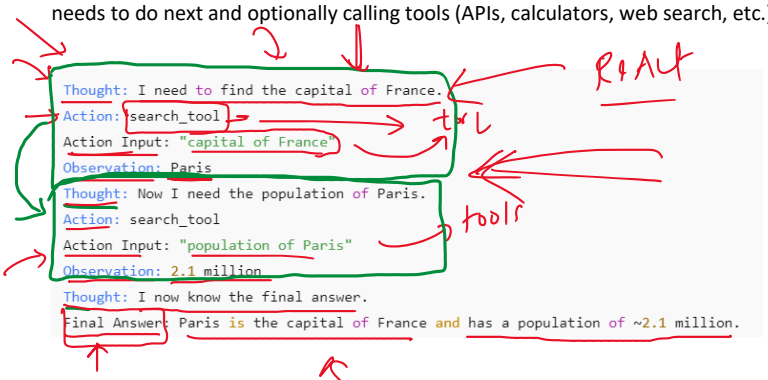
29 April 2025 22:53

1. ReAct

01 May 2025 14:17

ReAct 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 an answer in one go, the model thinks step by step, deciding what it needs to do next and optionally calling tools (APIs, calculators, web search, etc.) to help it.



ReAct is useful for:

- Multi-step problems
- Tool-augmented tasks (web search, database lookup, etc.)
- Making the agent's reasoning **transparent and auditable**

It was first introduced in the paper:

"*ReAct: Synergizing Reasoning and Acting in Language Models*" (Yao et al., 2022)

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao[*,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

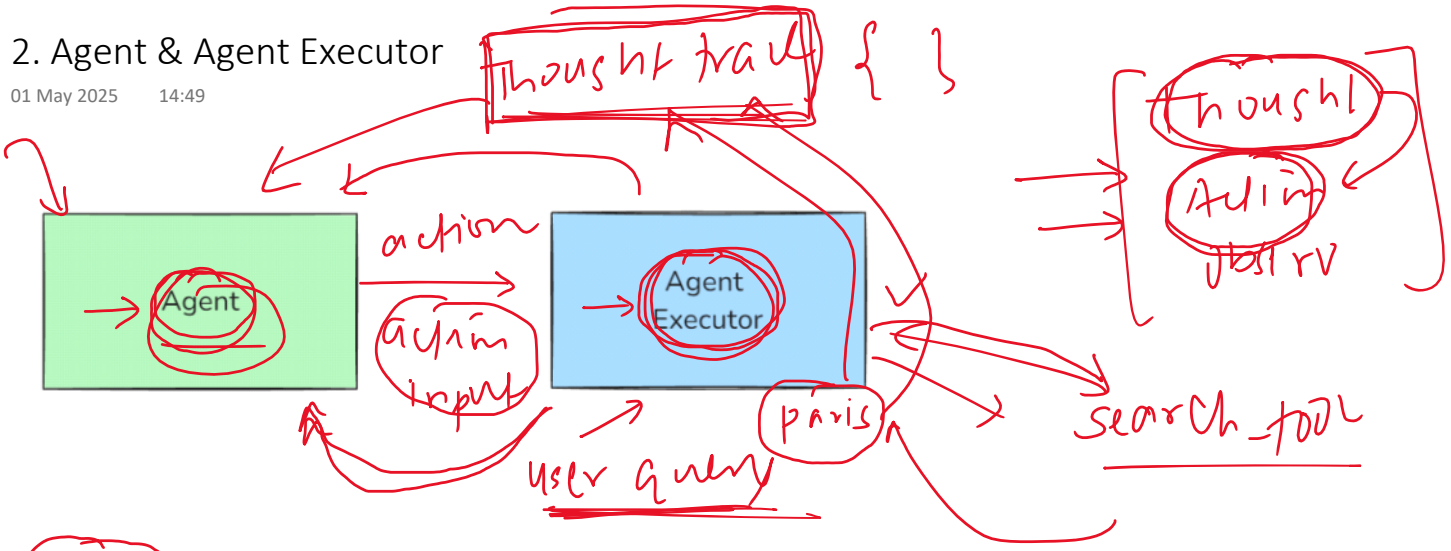
[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

2. Agent & Agent Executor

01 May 2025 14:49



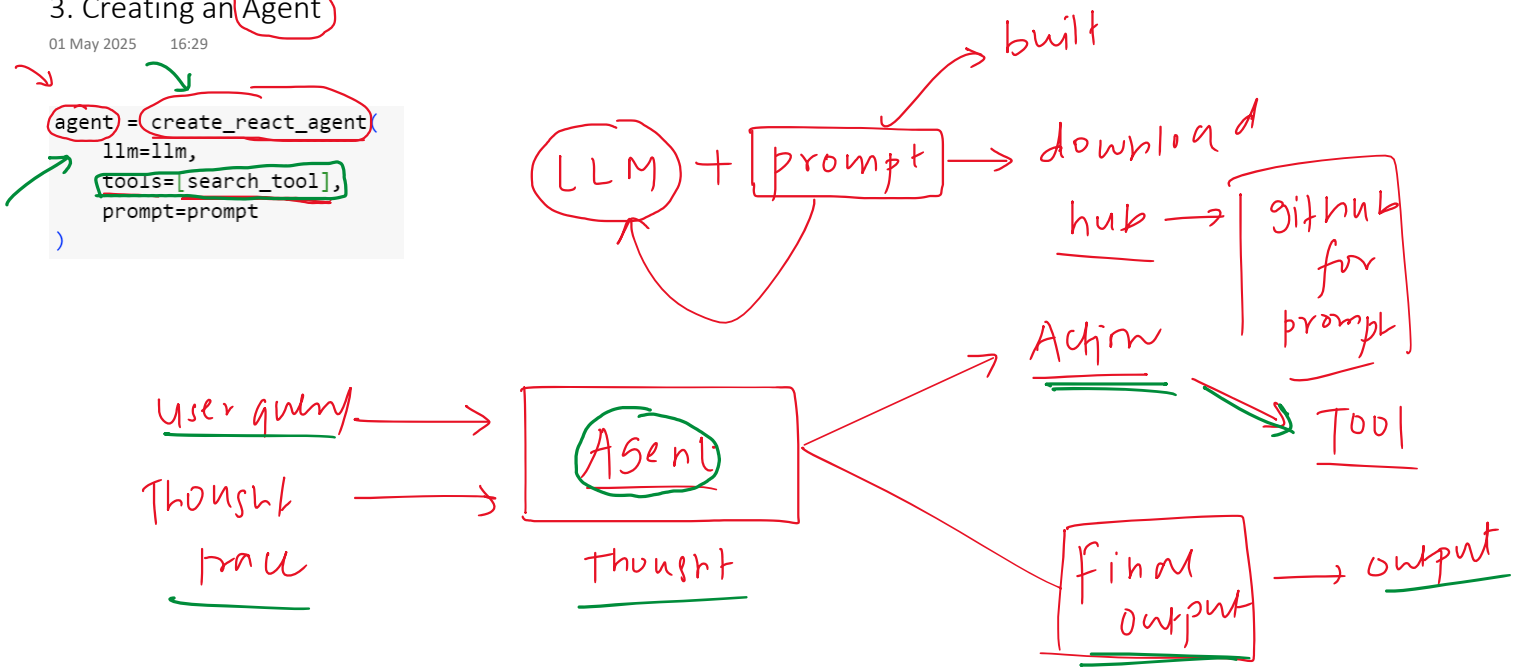
`AgentExecutor` orchestrates the **entire loop**:

1. Sends inputs and previous messages to the agent
2. Gets the next `action` from agent
3. Executes that tool with provided input
4. Adds the tool's `observation` back into the history
5. Loops again with updated history until the agent says `Final Answer`.

3. Creating an Agent

01 May 2025 16:29

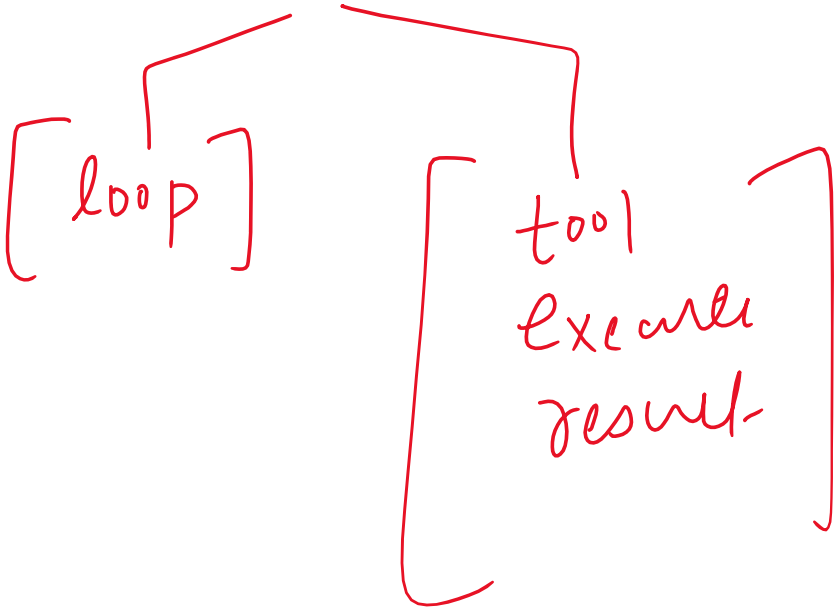
```
agent = create_react_agent(
    llm=llm,
    tools=[search_tool],
    prompt=prompt
)
```



4. Creating an Agent Executor

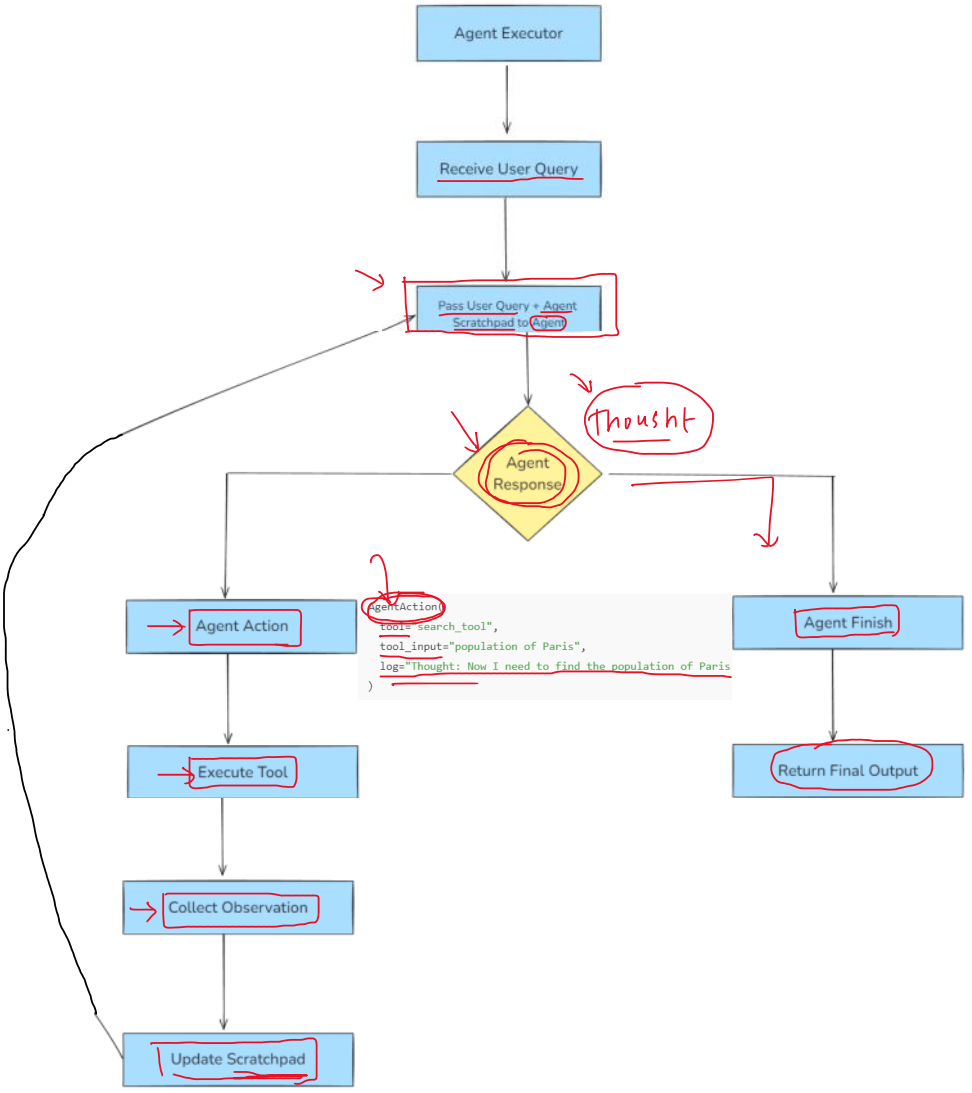
01 May 2025 16:30

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



5. Flow Chart

01 May 2025 16:30



6. Example

01 May 2025 17:23

🧠 Input Query:

> "What is the capital of France and what is its population?"

```
Answer the following questions as best you can. You have access to the following tools:

search_tool: Useful for answering general knowledge questions by querying a search API.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [search_tool]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: What is the capital of France and what is its population?
Thought:
```

```
Thought: I need to find the capital of France first.
Action: search_tool
Action Input: "capital of France"
```

```
AgentAction(
  tool="search_tool",
  tool_input="capital of France",
  log="Thought: I need to find the capital of France first.
)
```

```
observation = search_tool("capital of France")
```

→ `"Paris is the capital of France."`

```
Thought: I need to find the capital of France first.
Action: search_tool
Action Input: "capital of France"
Observation: Paris is the capital of France.
```

📌 Current state so far:

- **User Input**:

`What is the capital of France and what is its population?`

- **Agent Scratchpad after Step 2**:

text

```
Thought: I need to find the capital of France first.
Action: search_tool
Action Input: "capital of France"
Observation: Paris is the capital of France.
```

```
Answer the following questions as best you can. You have access to the following tools

search_tool: Useful for answering general knowledge questions by querying a search API.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [search_tool]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: What is the capital of France and what is its population?
Thought: I need to find the capital of France first.
Action: search_tool
Action Input: "capital of France"
Observation: Paris is the capital of France.
Thought:
```

```
Thought: Now I need to find the population of Paris.
Action: search_tool
Action Input: "population of Paris"
```

```
AgentAction(
  tool="search_tool",
  tool_input="population of Paris",
```

```
AgentAction(
  tool="search_tool",
  tool_input="population of Paris",
  log="Thought: Now I need to find the population of Paris.
)
```

```
observation = search_tool("population of Paris")
```

```
"Paris has a population of approximately 2.1 million."
```

```
Thought: I need to find the capital of France first.
Action: search_tool
Action Input: "capital of France"
Observation: Paris is the capital of France.
Thought: Now I need to find the population of Paris.
Action: search_tool
Action Input: "population of Paris"
Observation: Paris has a population of approximately 2.1 million.
```

```
Answer the following questions as best you can. You have access to the following tools:

search_tool: Useful for answering general knowledge questions by querying a search API.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [search_tool]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: What is the capital of France and what is its population?
Thought: I need to find the capital of France first.
Action: search_tool
Action Input: "capital of France"
Observation: Paris is the capital of France.
Thought: Now I need to find the population of Paris.
Action: search_tool
Action Input: "population of Paris"
Observation: Paris has a population of approximately 2.1 million.
Thought:
```

```
Thought: I now know the final answer.
Final Answer: Paris is the capital of France and has a population of approximately 2.1 million.
```

```
Thought: I now know the final answer.
Final Answer: Paris is the capital of France and has a population of approximately 2.1 million.
```

```
AgentFinish(
  return_values={"output": "The capital of France is Paris..."},
  log="Thought: I now know the final answer.\nFinal Answer: ..."
)
```

```
{
  "output": "Paris is the capital of France and has a population of approximately 2.1 million."
}
```