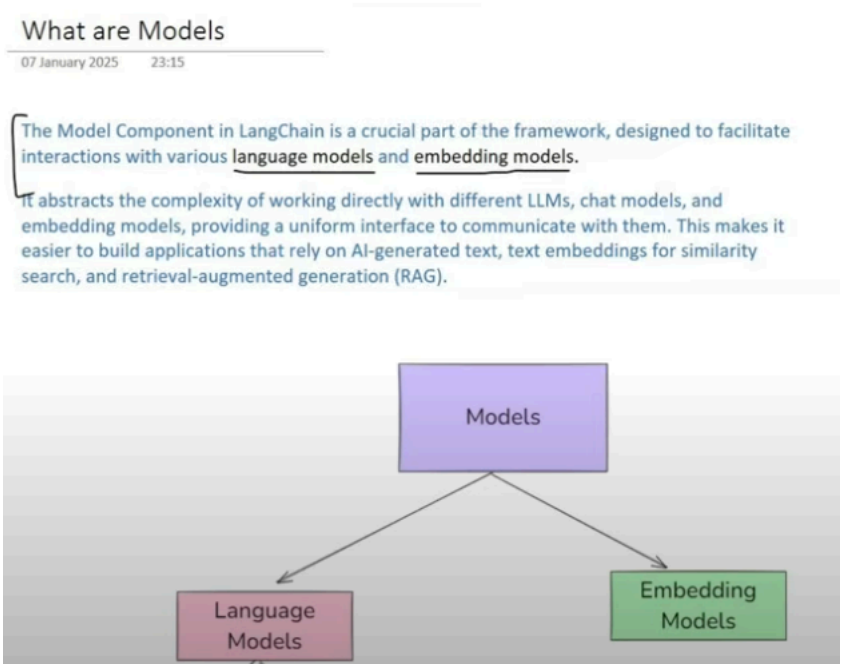
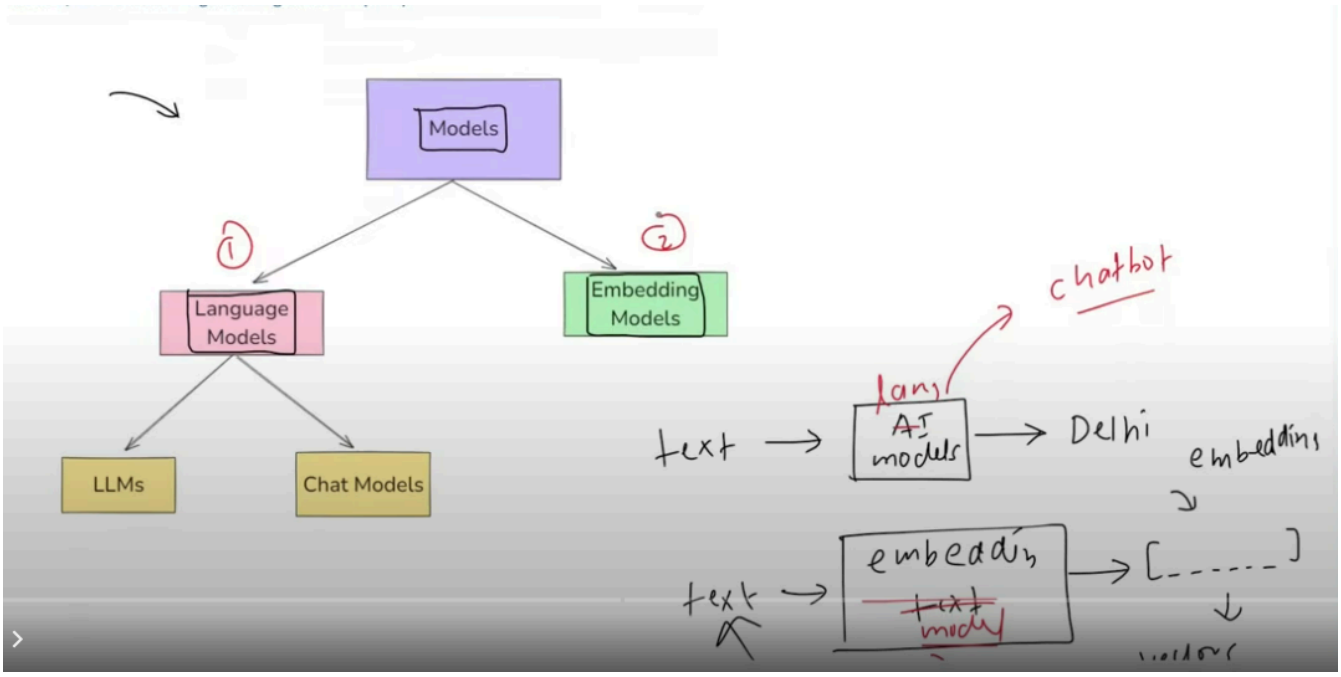
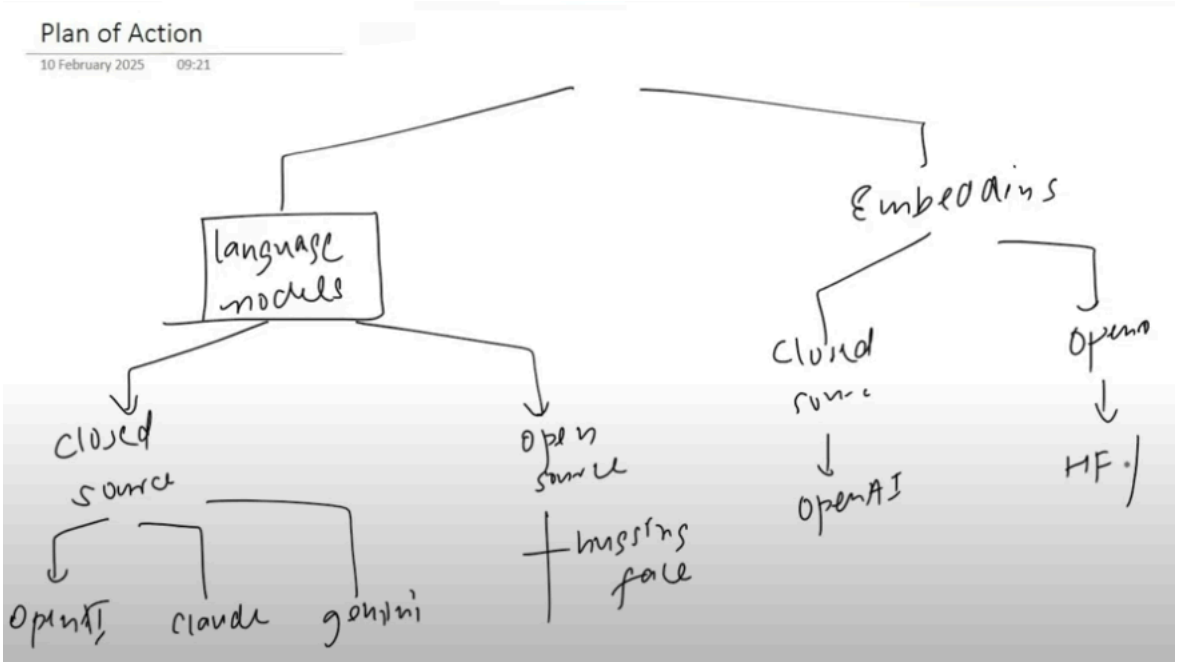
Start coding or generate with AI.

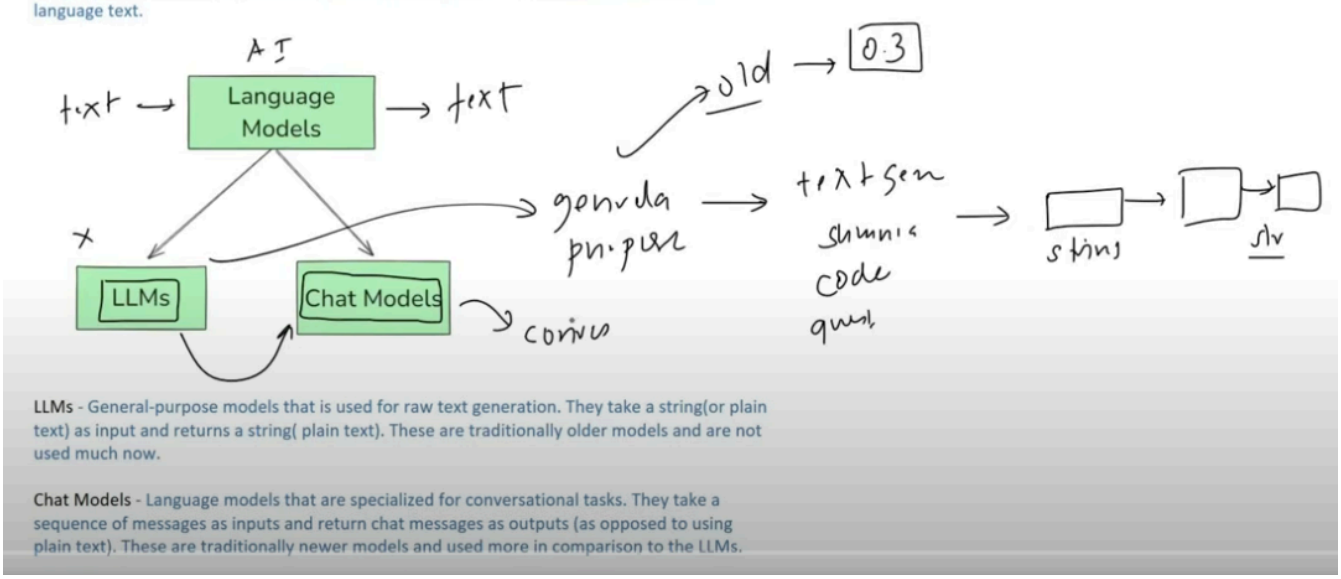
Model Component







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.

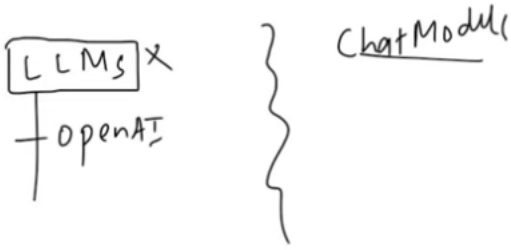
Chat Models - Language models that are specialized for conversational tasks. They take a sequence of messages as inputs and return chat messages as outputs (as opposed to using plain text). These are traditionally newer models and used more in comparison to the LLMs.

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.

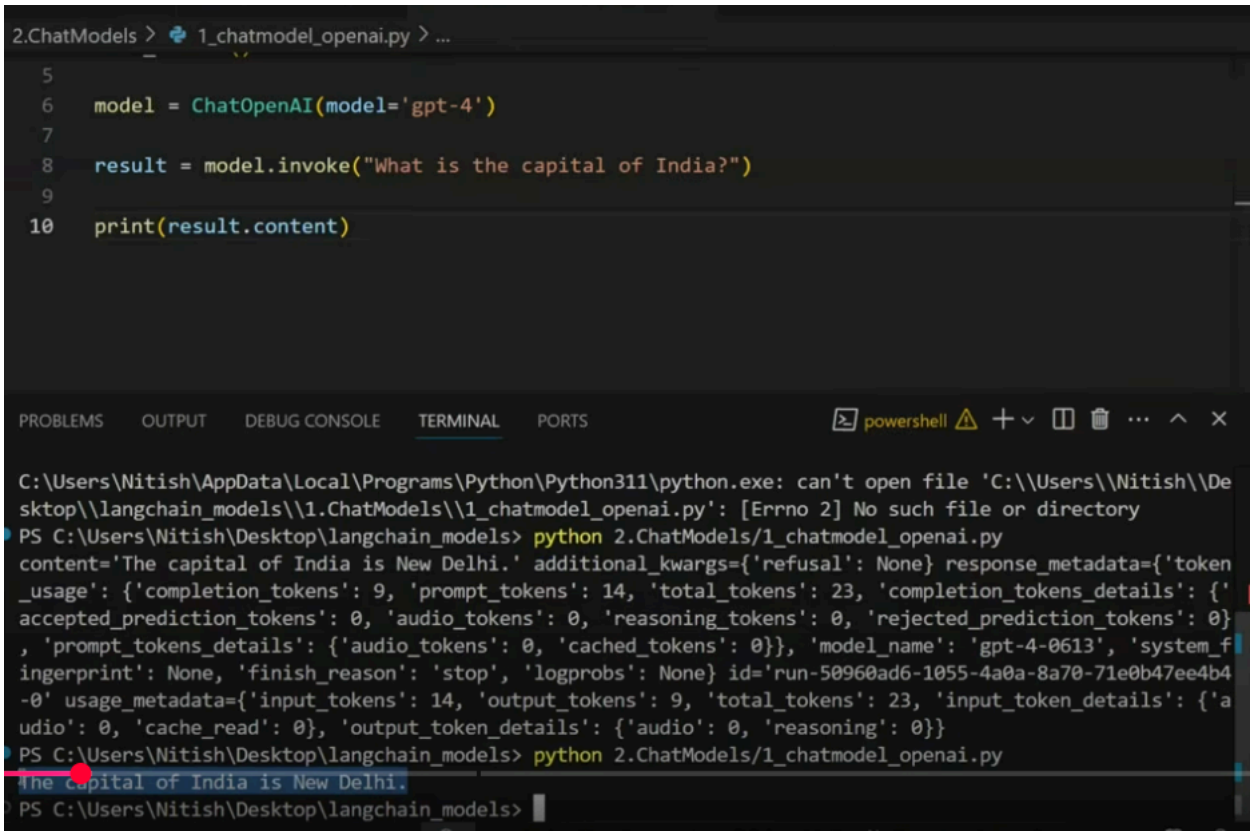
Feature	LLMs (Base Models)	Chat Models (Instruction-Tuned)
Purpose	Free-form text generation	Optimized for multi-turn conversations
Training Data	General text corpora (books, articles)	Fine-tuned on chat datasets (dialogues, user-assistant conversations)
Memory & Context	No built-in memory	Supports structured 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

Demo

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



1. platform.openai.com
2. console.anthropic.com
3. ai.google.dev
4. huggingface.co

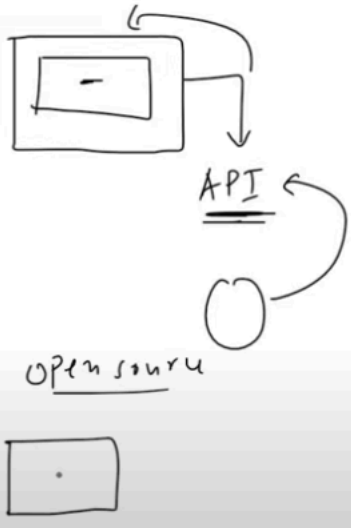


Open Source Models

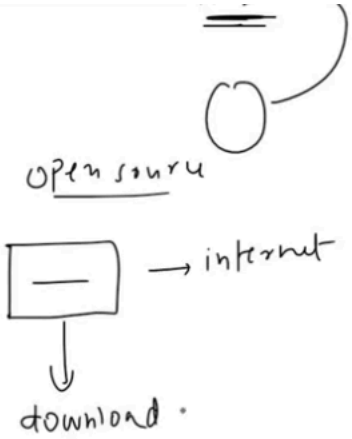
11 February 2025 08:58

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



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



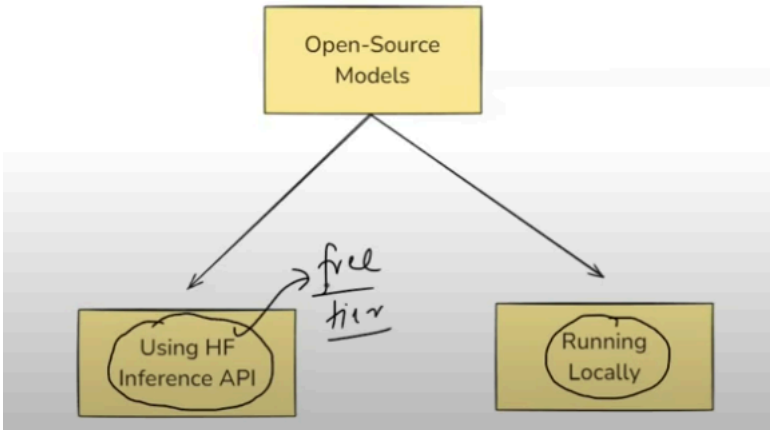
Some Famous Open Source Models

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?

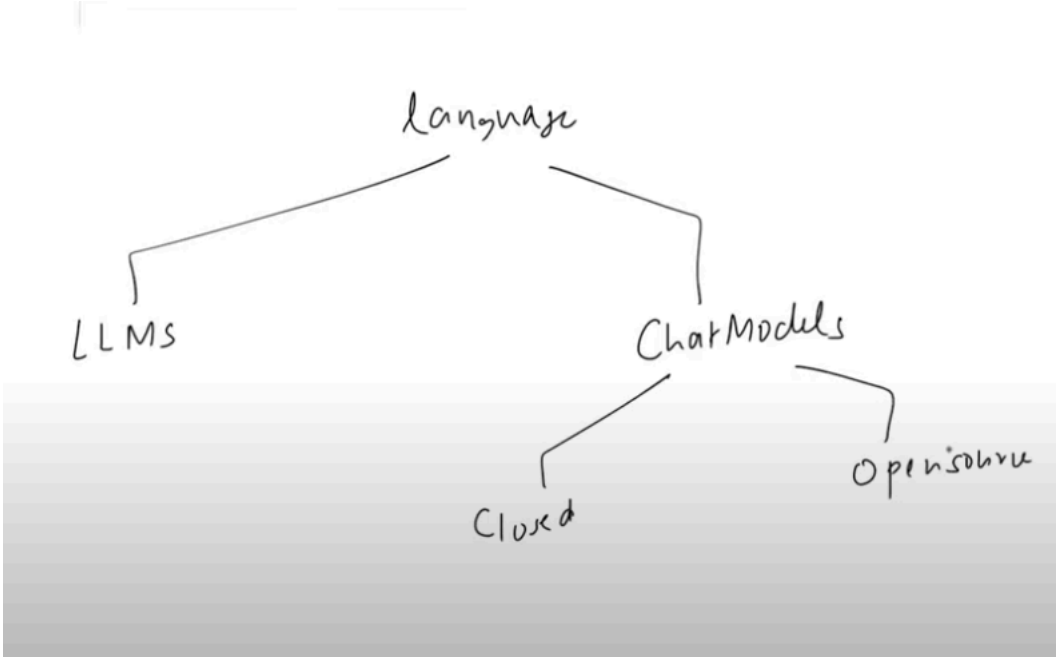
HuggingFace - The largest repository of open-source LLMs

Ways to use Open-source Models

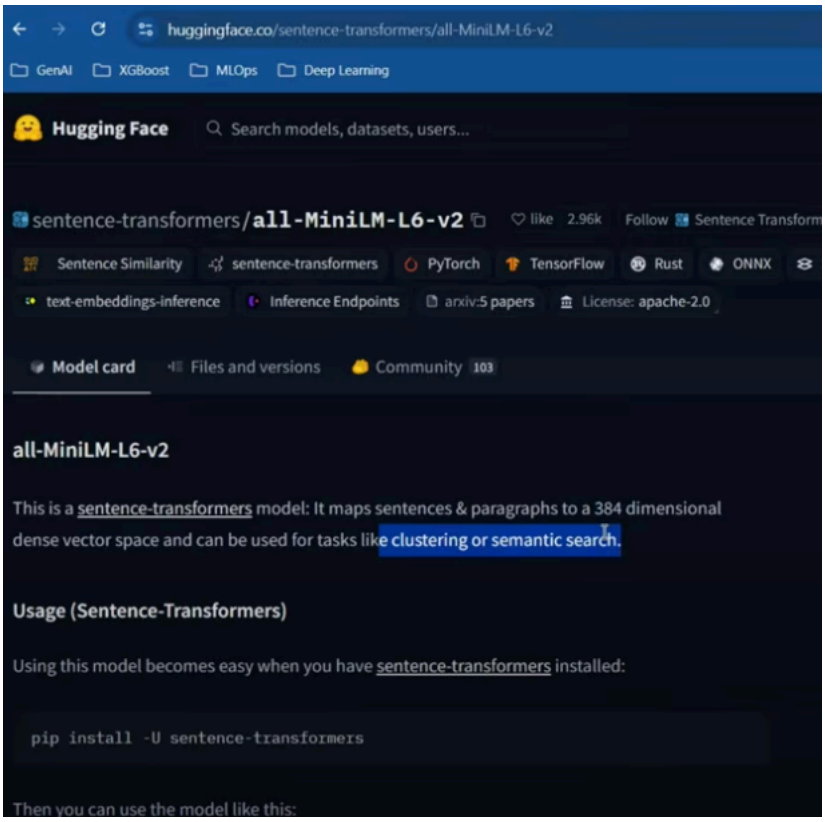


Disadvantages

Disadvantage	Details
High Hardware 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.



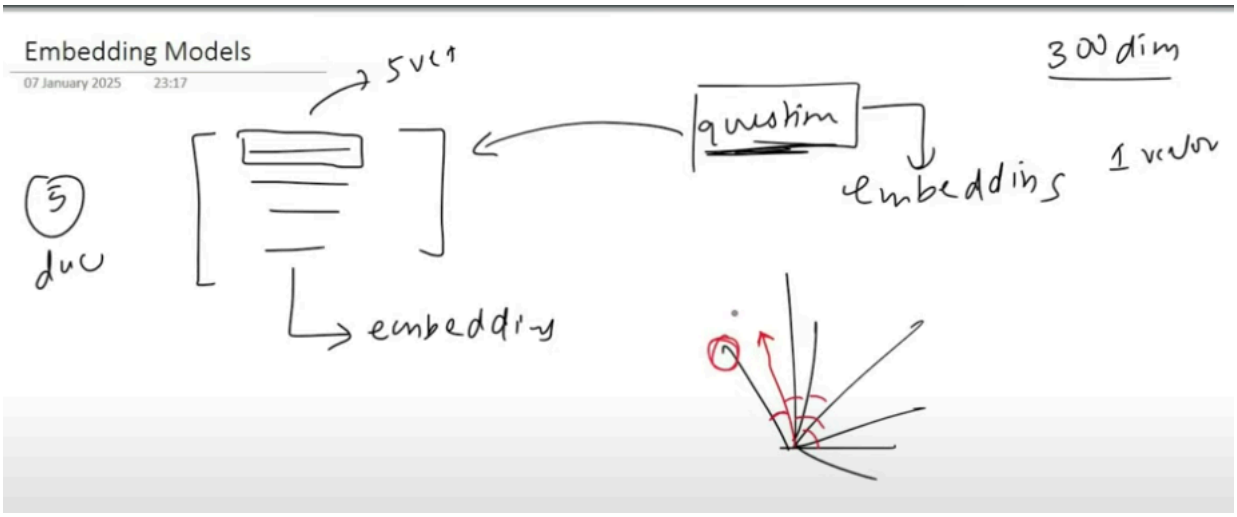
By default, the length of the embedding vector will be 1536 for `text-embedding-3-small` or 3072 for `text-embedding-3-large`. You can reduce the dimensions of the embedding by passing in the dimensions parameter without the embedding losing its concept-representing properties. We go into more detail on embedding dimensions in the embedding use case section.



1_embedding_openai_query.py | 2_embedding_openai_docs.py | 3_embedding_hf_local.py

3.EmbeddingModels > 3_embedding_hf_local.py > ...

```python
from langchain_huggingface import HuggingFaceEmbeddings

embedding = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L6-v2')

text = "Delhi is the capital of India"

vector = embedding.embed_query(text)

print(str(vector))
```



✅ If you set `temperature = 0`:

- The model behaves **deterministically**.
- **It always gives the same response** to the same prompt.
- It selects the **most likely next token** at every step — no randomness at all.

🔁 If you increase `temperature` (> 0):

- The model becomes **more random**.
- Responses **can vary** each time, even with the same prompt.
- Higher values like `0.7` or `1.0` introduce **creative or surprising outputs**, but possibly **less reliable**.

📌 Summary:

Temperature	Behavior	Response Variability
0	Deterministic	❌ No change
~0.3	Conservative	🔁 Slight variation
~0.7	Balanced	🔁 Noticeable variation
≥ 1.0	Creative/Random	🔁 Highly variable

🔢 `top_k`: Pick from the **top k most popular items**

Let's say the waiter **shows you only the k most popular dishes**.

🍛 Example:

If:

- The top 3 dishes are:
  1. Butter Chicken (30% popular)
  2. Paneer Tikka (20%)
  3. Veg Biryani (15%)

Then:

- `top_k = 3` → you'll **only choose from those 3**, ignoring the rest of the menu.
- If you **increase** `k` **to 10**, you have **more variety**, but also the risk of choosing something less loved (e.g., boiled okra 😖).

> 🔁 So `top_k` = "Give me the **top k best-sellers**, and I'll pick from those."

📊 `top_p`: Pick until you've covered **p% of the popularity**

Instead of a fixed number of dishes, you say:

> "Show me dishes until we've covered 90% of what people usually order."

This is `top_p` (nucleus sampling).

Let's say dish popularity is:

Dish	Popularity
Butter Chicken	30%
Paneer Tikka	20%
Veg Biryani	15%
Chole Bhature	10%
Dosa	5%
Others (small %)	...

To reach `top_p = 0.9` (90%), you include dishes until their cumulative probability ≥ 90%:

- Butter Chicken (30%)
- Paneer Tikka (20%)
- Veg Biryani (15%)
- Chole Bhature (10%)
- Dosa (5%)
  Total = 80%, add more until you reach ≥ 90%.

Then randomly pick a dish from those.

> ✅ `top_p` = "Keep adding items to my shortlist until I've seen enough options that 90% of people usually pick from."

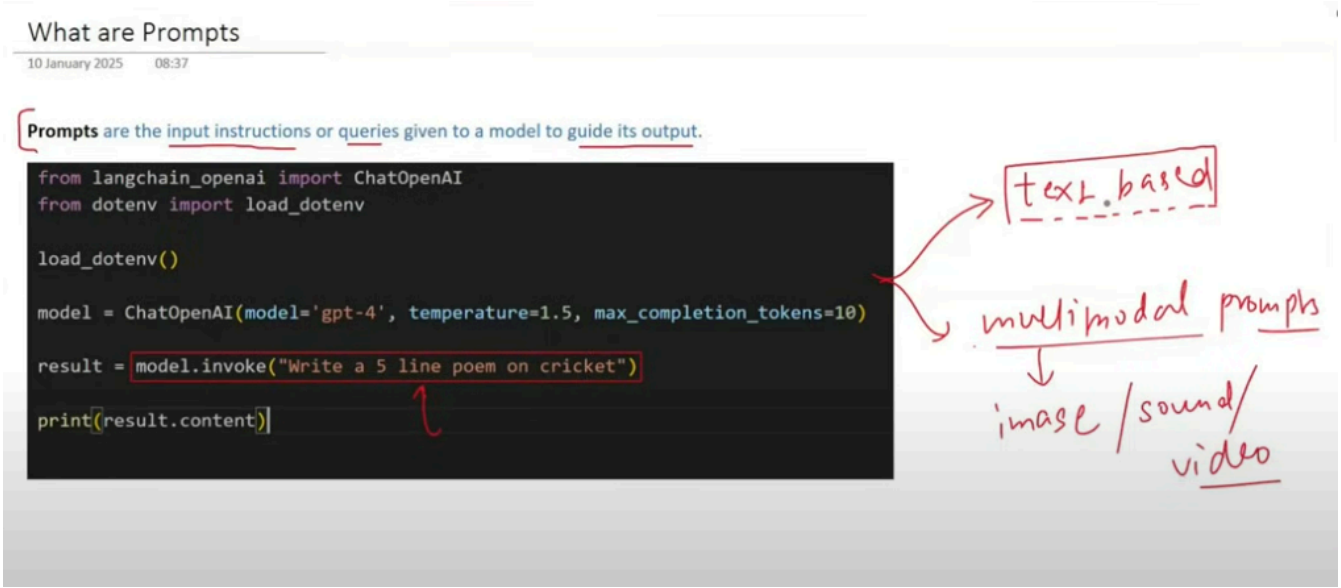
🧠 Summary: How They Differ

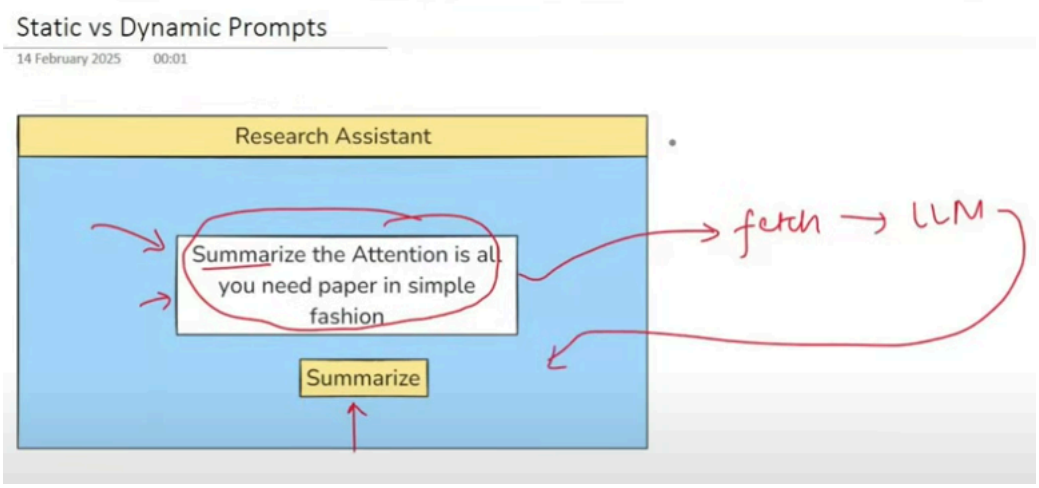
Parameter	How it Works	Restaurant Analogy
`top_k`	Fixed **number** of options	"Show me the top 5 best-selling dishes"
`top_p`	Dynamic **coverage threshold**	"Show me dishes until I've seen 90% of what's popular"

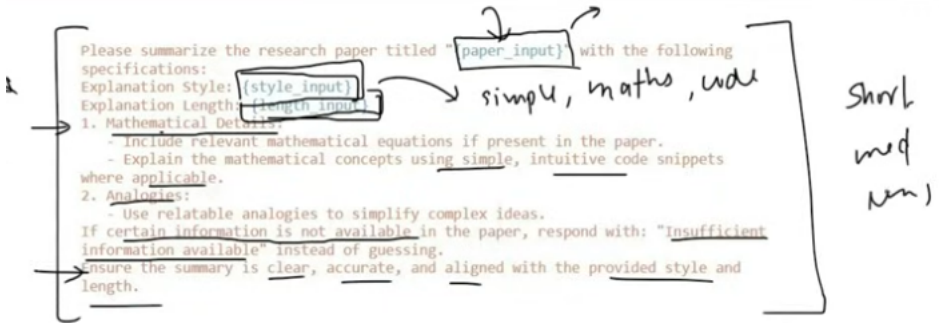
⚖️ Real-World Use

- `top_k` is **fixed-width** filtering: best for when you want consistent shortlist.
- `top_p` is **flexible** and adapts to shape of probability distribution — good for naturally diverse but coherent outputs.

Prompts







📌 Static Prompts

A **static prompt** is a **fixed, hardcoded text input** given to the model.

✅ **Characteristics:**

- Does **not change at runtime**
- Often used for **simple tasks** or when you want **reproducible output**
- Easier to test/debug

📄 **Example:**

```python
prompt = "Explain what a neural network is."
```

🔄 Dynamic Prompts

A **dynamic prompt** is **generated or modified at runtime** using variables, user inputs, or external data.

✅ **Characteristics:**

- Customizes the prompt based on context
- Essential for **chatbots, data-driven tasks, multi-step chains**, etc.
- Enables **personalization** and **flexibility**

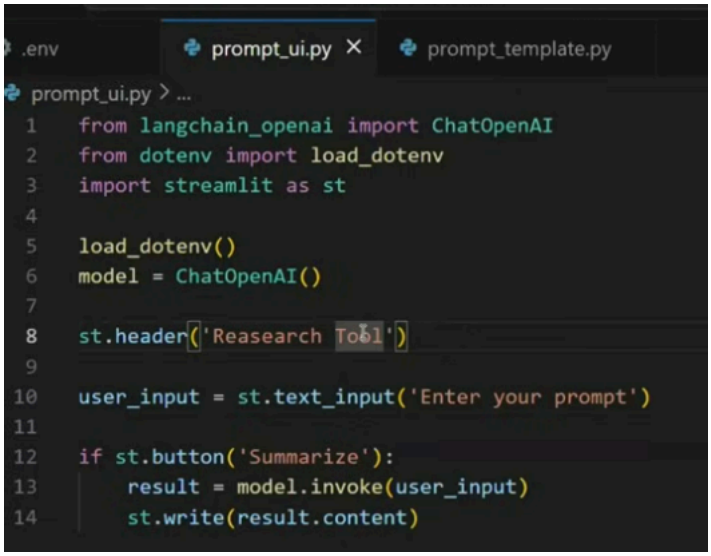
📄 **Example:**

```python
user_question = "What is backpropagation?"
prompt = f"Explain the concept of {user_question} in simple terms."
```

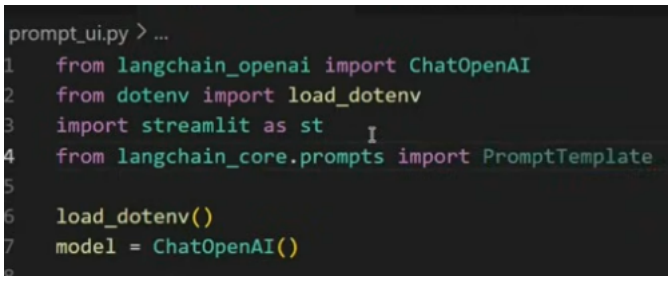
🔍 Key Differences:

Feature	Static Prompt	Dynamic Prompt
Customizability	Fixed content	Varies with inputs or logic
Flexibility	Limited	High
Use cases	Testing, documentation, templates	Chatbots, LangChain chains, personalized tasks
Debuggability	Easier (no variables)	Requires more validation

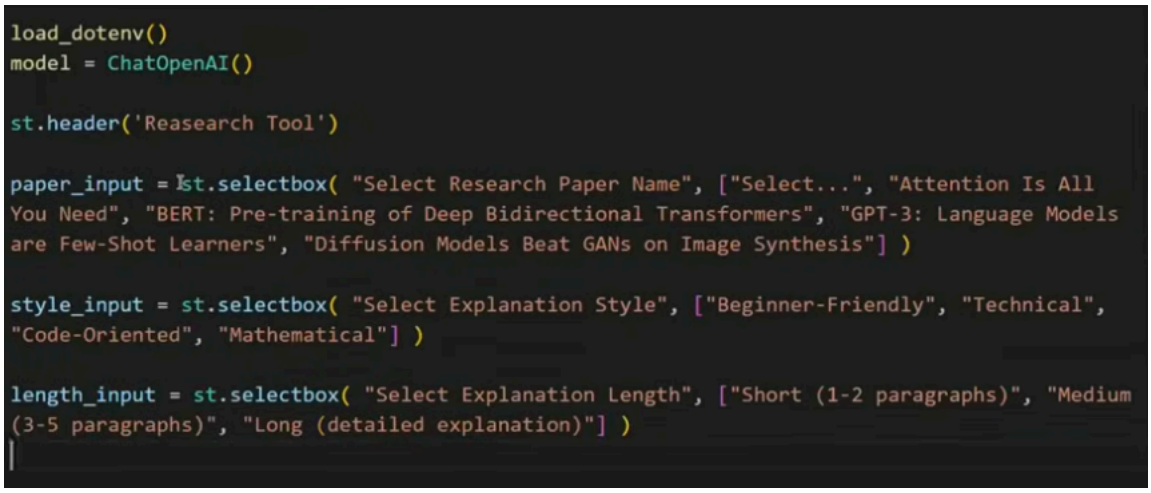
Dynamic prompt code



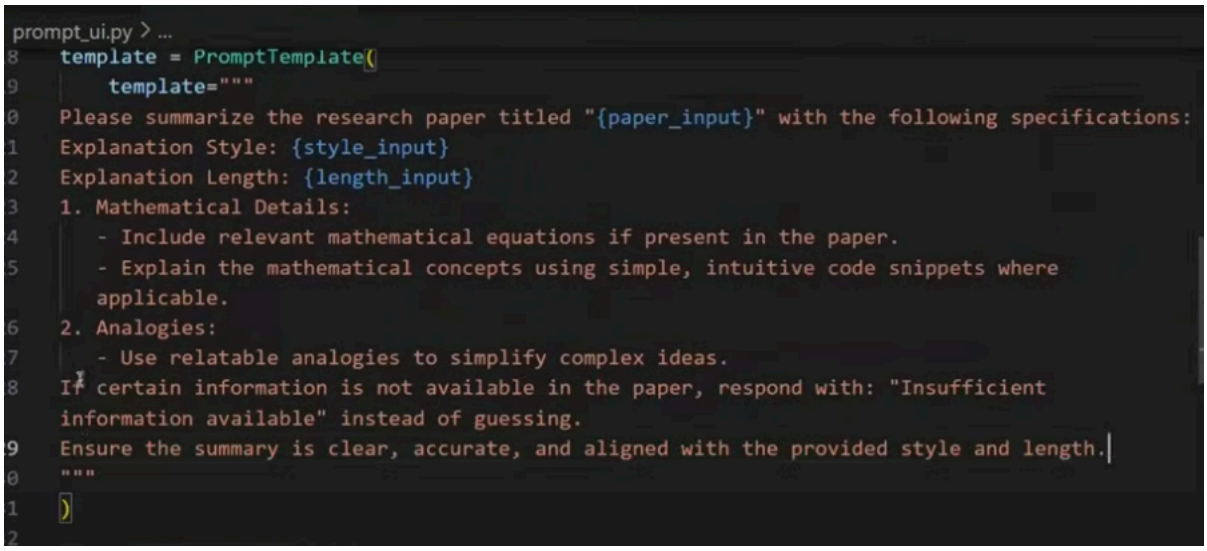
```
.env   prompt_ui.py ×   prompt_template.py
prompt_ui.py > ...
1   from langchain_openai import ChatOpenAI
2   from dotenv import load_dotenv
3   import streamlit as st
4
5   load_dotenv()
6   model = ChatOpenAI()
7
8   st.header('Reasearch Tool')
9
10  user_input = st.text_input('Enter your prompt')
11
12  if st.button('Summarize'):
13      result = model.invoke(user_input)
14      st.write(result.content)
```



```
prompt_ui.py > ...
1   from langchain_openai import ChatOpenAI
2   from dotenv import load_dotenv
3   import streamlit as st
4   from langchain_core.prompts import PromptTemplate
5
6   load_dotenv()
7   model = ChatOpenAI()
```



```
load_dotenv()
model = ChatOpenAI()

st.header('Reasearch Tool')

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)"] )
```



```
prompt_ui.py > ...
8   template = PromptTemplate(
9       template="""
0   Please summarize the research paper titled "{paper_input}" with the following specifications:
1   Explanation Style: {style_input}
2   Explanation Length: {length_input}
3   1. Mathematical Details:
4       - Include relevant mathematical equations if present in the paper.
5       - Explain the mathematical concepts using simple, intuitive code snippets where
        applicable.
6   2. Analogies:
7       - Use relatable analogies to simplify complex ideas.
8   If certain information is not available in the paper, respond with: "Insufficient
    information available" instead of guessing.
9   Ensure the summary is clear, accurate, and aligned with the provided style and length.
0   """
1   )
2
```

```
    applicable.
  2. Analogies:
     - Use relatable analogies to simplify complex ideas.
  If certain information is not available in the paper, respond with: "Insufficient
  information available" instead of guessing.
  Ensure the summary is clear, accurate, and aligned with the provided style and length.
  """,
  input_variables=['paper_input', 'style_input','length_input']
  )
```

```
# fill the placeholders
prompt = template.invoke({
    'paper_input':paper_input,
    'style_input':style_input,
    'length_input':length_input
})

if st.button('Summarize'):
    result = model.invoke(prompt)
    st(result.content)
```

prompt_ui.py > ...

```
0   Please summarize the research paper titled "{paper_input}" with the following specifications:
1   Explanation Style: {style_input}
2   Explanation Length: {length_input}
3   1. Mathematical Details:
4      - Include relevant mathematical equations if present in the paper.
5      - Explain the mathematical concepts using simple, intuitive code snippets where
       applicable.
6   2. Analogies:
7      - Use relatable analogies to simplify complex ideas.
8   If certain information is not available in the paper, respond with: "Insufficient
    information available" instead of guessing.
9   Ensure the summary is clear, accurate, and aligned with the provided style and length.
0   """,
1   input_variables=['paper_input', 'style_input'],
2   validate_template=True
3   )
```

```
   information available" instead of guessing.
   Ensure the summary is clear, accurate, and aligned with the provided style and length.
   """,
   input_variables=['paper_input', 'style_input'],
   validate_template=True
   )

   # fill the placeholders
   prompt = template.invoke({
```

BLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS streamlit

```
ile "C:\Users\Nitish\Desktop\langchain_prompts\prompt_ui.py", line 18, in <module>
 template = PromptTemplate(
            ^^^^^^^^^^^^^^
ile "C:\Users\Nitish\AppData\Local\Programs\Python\Python311\Lib\site-packages\langchain_core\load\seri
zable.py", line 125, in __init__
 super().__init__(*args, **kwargs)
ile "C:\Users\Nitish\AppData\Local\Programs\Python\Python311\Lib\site-packages\pydantic\main.py", line
, in __init__
 validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
antic_core._pydantic_core.ValidationError: 1 validation error for PromptTemplate
alue error, Invalid prompt schema; check for mismatched or missing input parameters from ['paper_input'
style_input']. [type=value_error, input_value={'template': '\nPlease su...'partial_variables': {}}, inp
```

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?

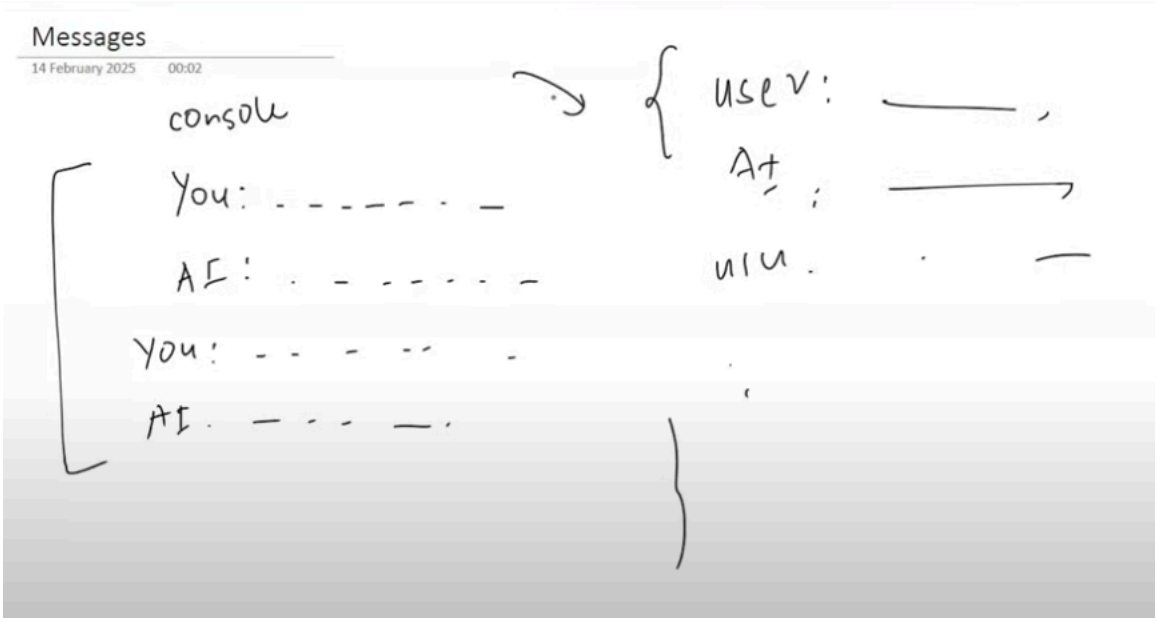
1. Default validation
2. Reusable
3. LangChain Ecosystem

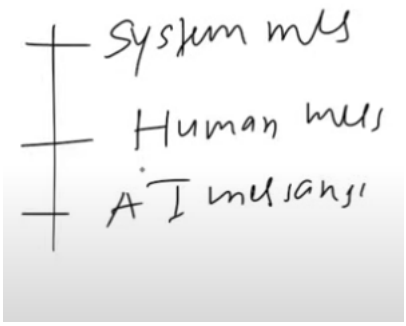
chatbot.py > ...

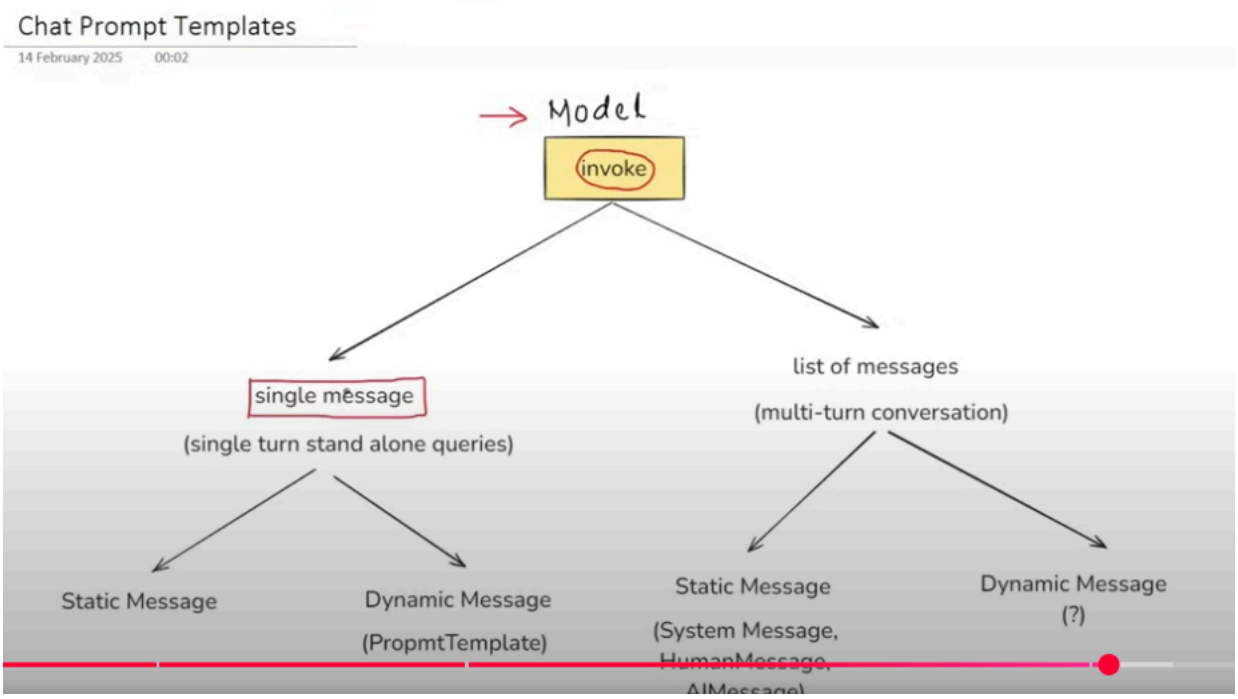
```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

model = ChatOpenAI()

while True:
    user_input = input('You: ')
    if user_input == 'exit':
        break
    result = model.invoke(user_input)
    print("AI: ",result.content)
```

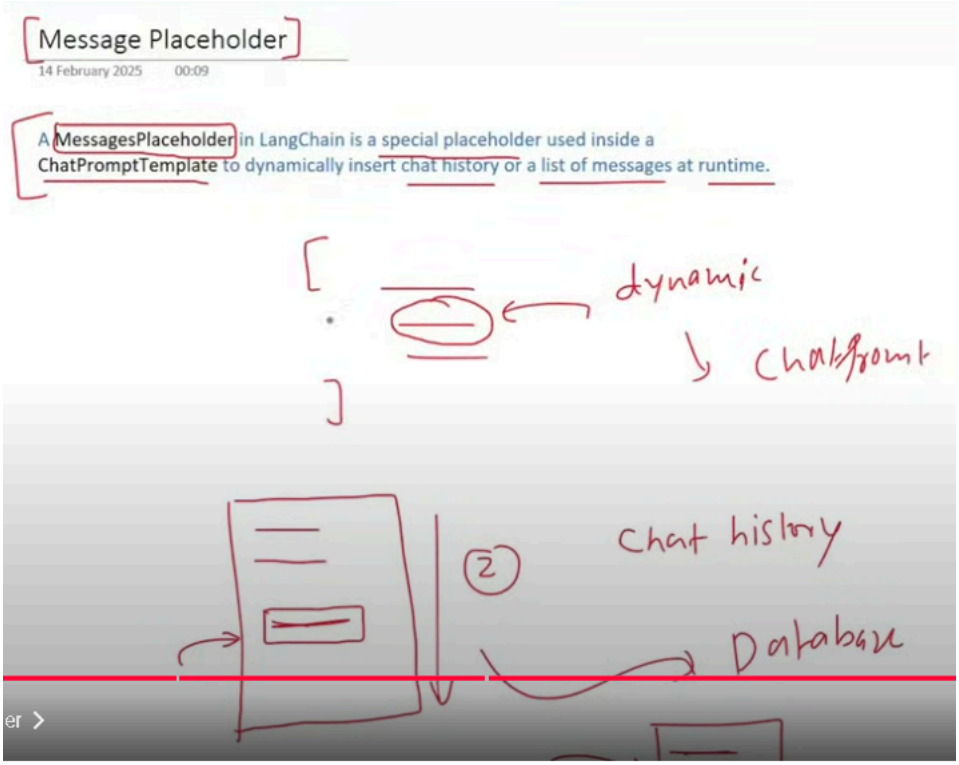
chatbot.py > ...

```python
chat_history = []

while True:
    user_input = input('You: ')
    chat_history.append(user_input)
    if user_input == 'exit':
        break
    result = model.invoke(chat_history)
    chat_history.append(result.content)
    print("AI: ",result.content)

print(chat_history)
```

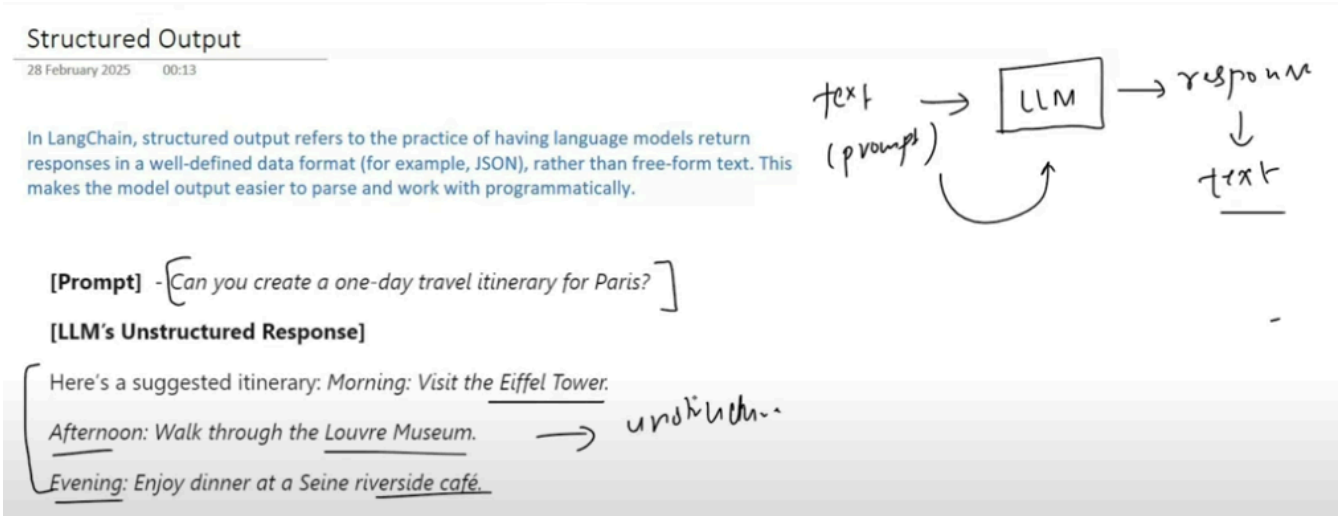

Messages
14 February 2025 00:02
console
You:
AI:
You:
AI.
user:
At
uiu.


System mls
Human mls
AI mlsages


Chat Prompt Templates
14 February 2025 00:02
Model
invoke
single message
(single turn stand alone queries)
list of messages
(multi-turn conversation)
Static Message
Dynamic Message
(PropmtTemplate)
Static Message
(System Message,
Dynamic Message
(?)

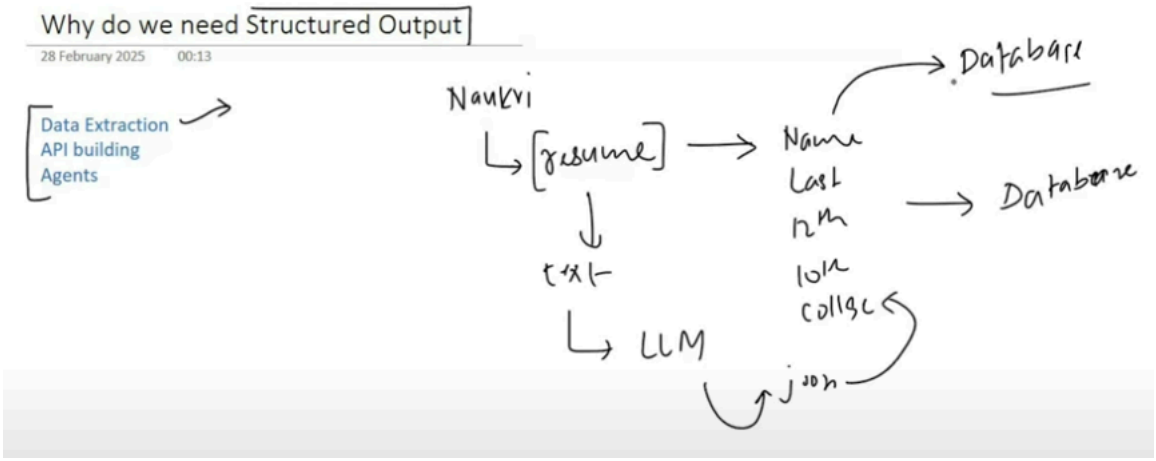


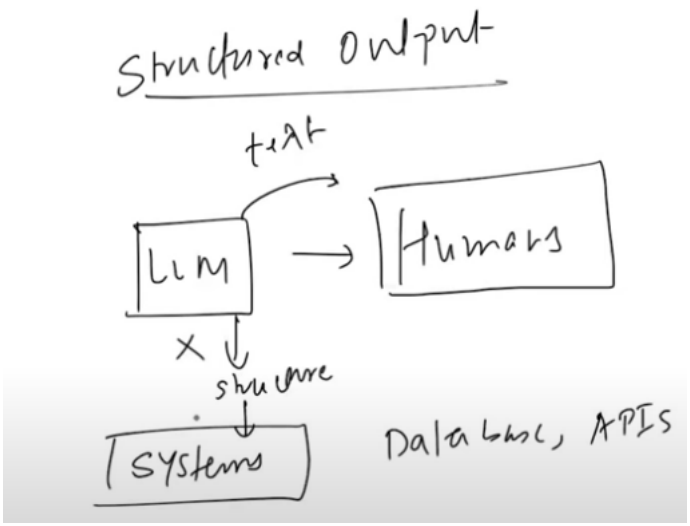
Structured output

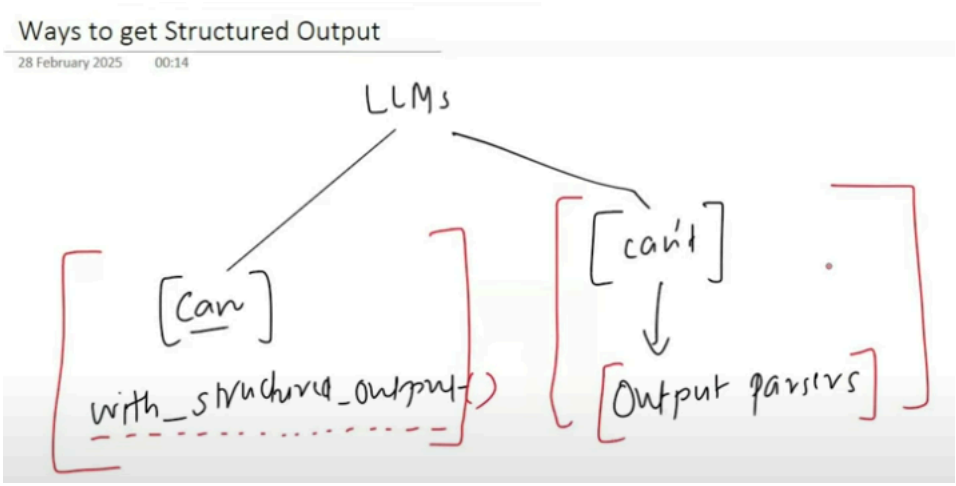


[JSON enforced output]

```
[
  {"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"}
]
```


Why do we need Structured Output
28 February 2025 00:13
Data Extraction
API building
Agents
Naukri
resume
text
LLM
Name
Last
Database
college
json


Structured Output
text
LLM
Humans
X
structure
Systems
Database, APIs

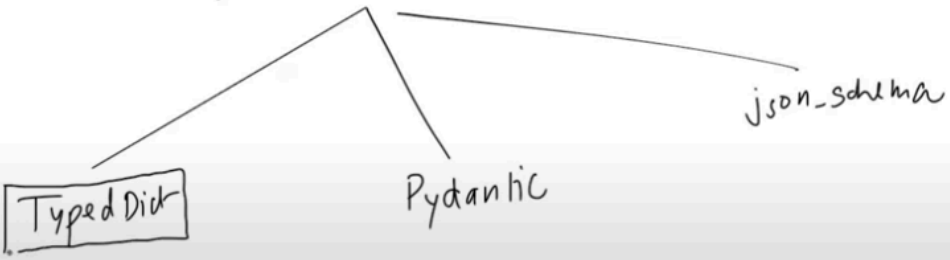

Ways to get Structured Output
28 February 2025 00:14
LLMs
Can
with_structured_output()
can't
Output parsers

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

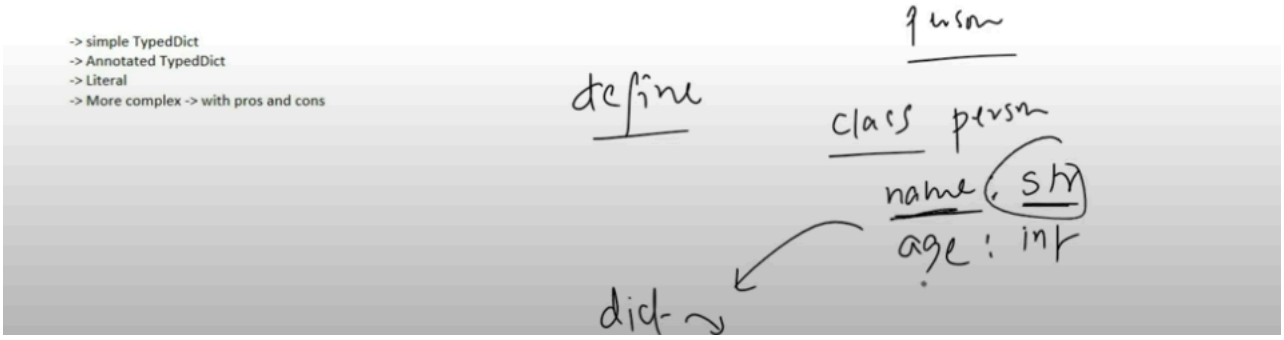
TypedDict is a way to define a dictionary in Python where you specify what keys and values should exist. It helps ensure that your dictionary follows a specific structure.

person = { name : nilish
age : 35 → str
}

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.

Basic example
Default values
Optional fields
Coerce
Builtin validation
Field Function -> default values, constraints, description, regex expressions
Returns pydantic object -> convert to json/dict

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:

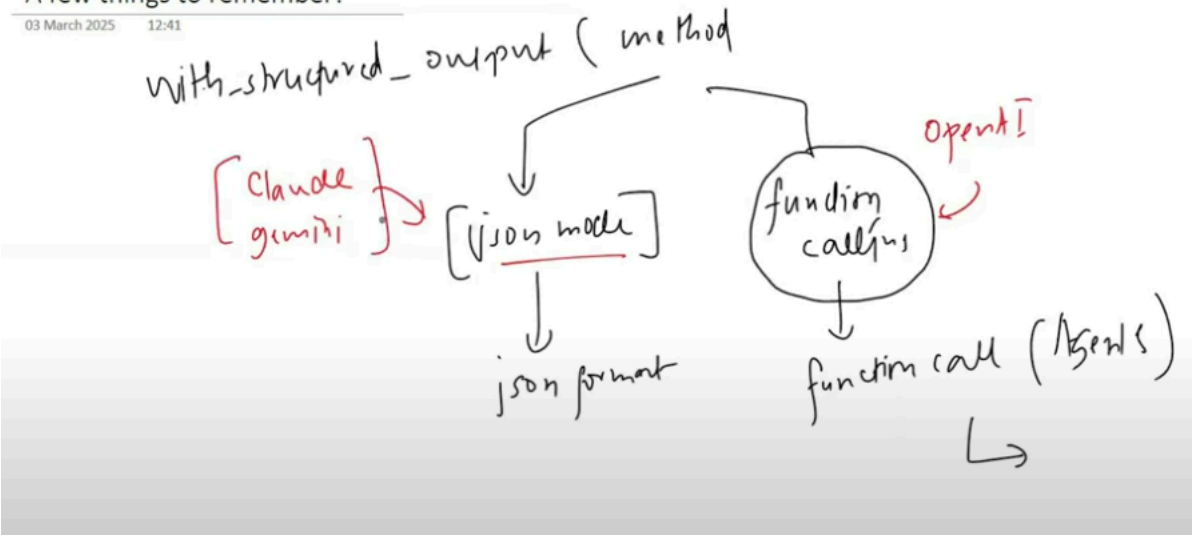
- You **don't want to import extra Python libraries** (Pydantic).
- You need **validation but don't need Python objects**.
- You want to define structure **in a standard JSON format**.

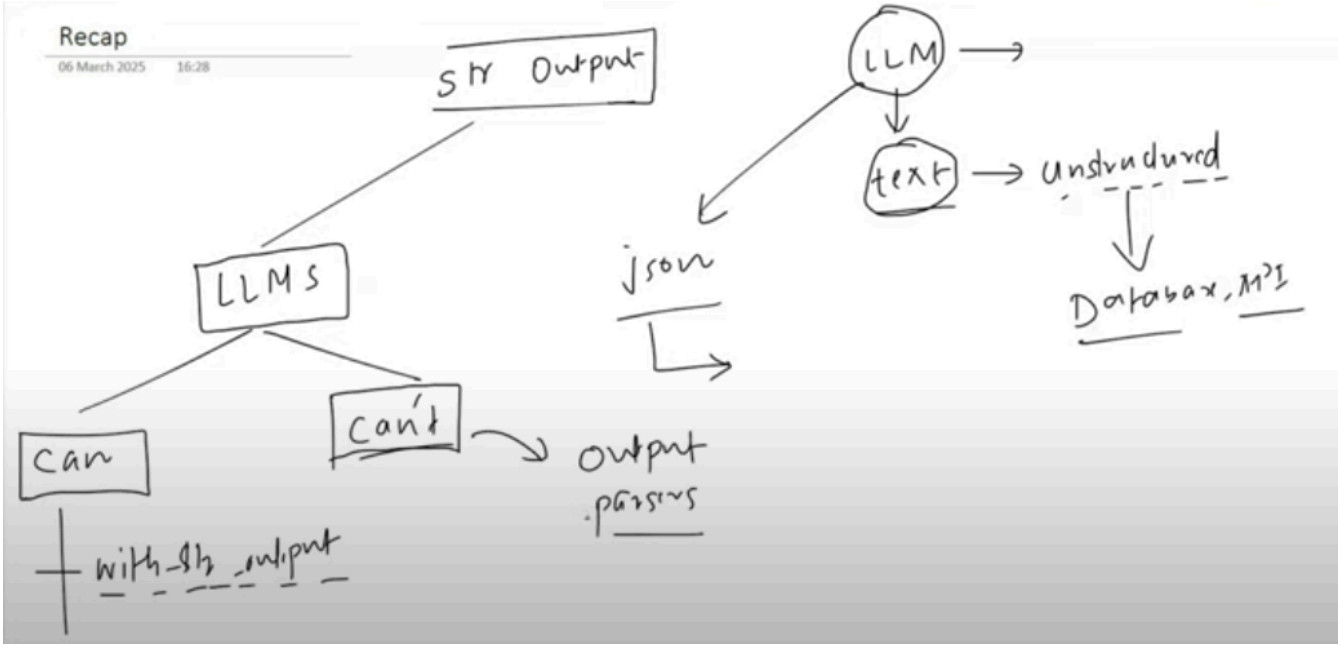
When to Use What?

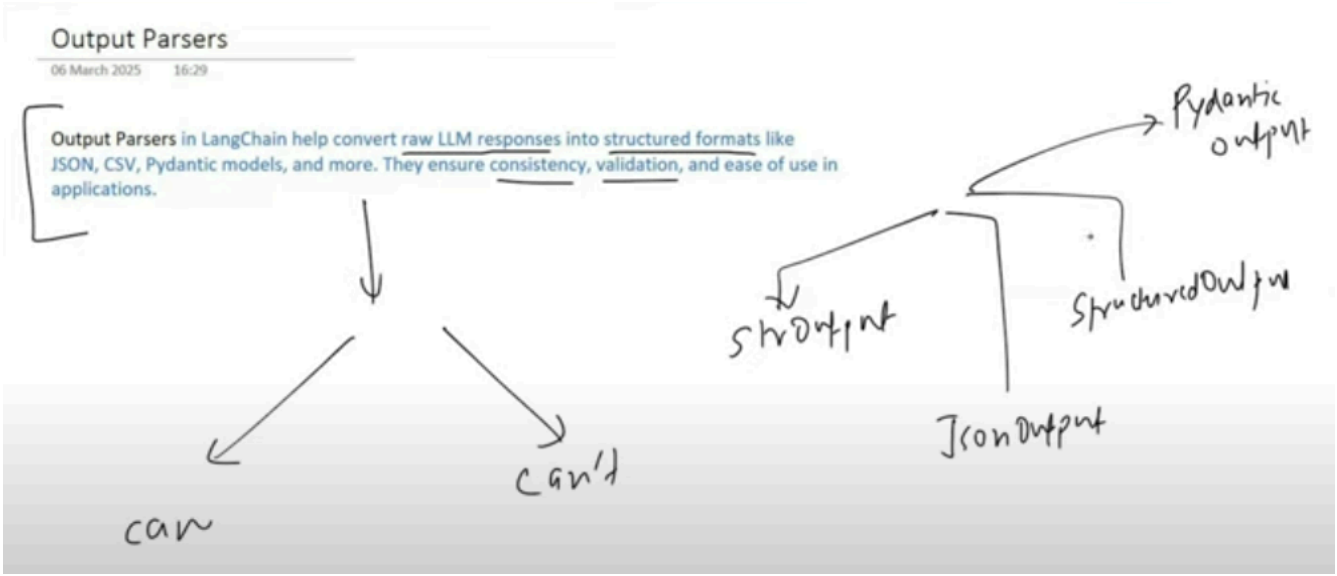
Feature	TypedDict	Pydantic	JSON Schema
Basic structure	✅	✅	✅
Type enforcement	✅	✅	✅
Data validation	❌	✅	✅
Default values	❌	✅	❌
Automatic conversion	❌	✅	❌
Cross-language compatibility	❌	❌	✅

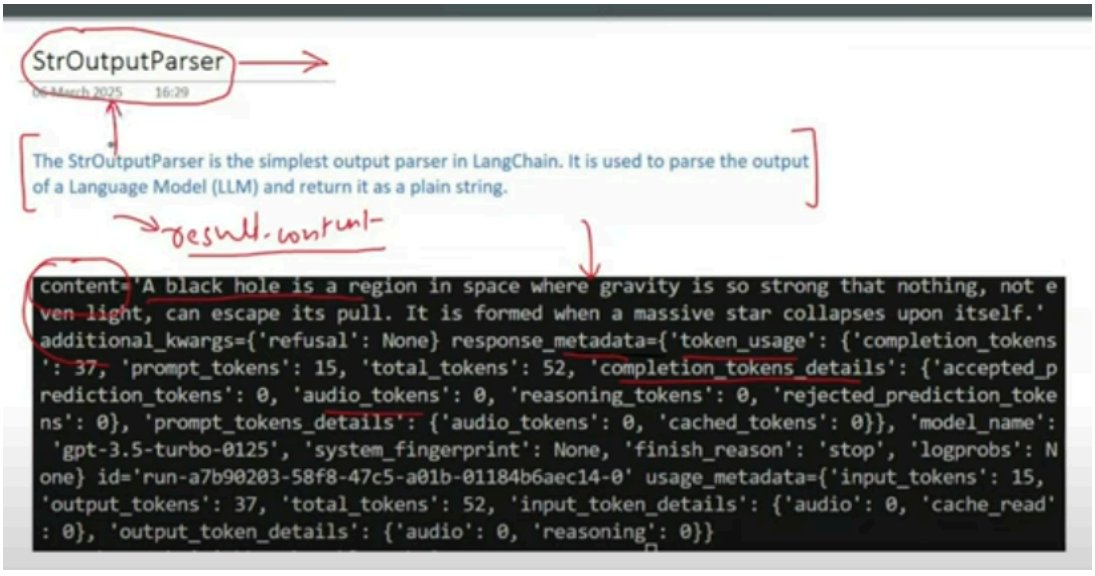
A few things to remember!

03 March 2025 12:41




Recap
06 March 2025 16:28
Str Output
LLM
text
unstructured
LLMS
json
Database, MDI
can
can't
output parsers
with_structured_output


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.
Pydantic output
StrOutput
StructuredOutput
JsonOutput
can
can't


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 even 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_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-3.5-turbo-0125', 'system_fingerprint': None, 'finish_reason': 'stop', 'logprobs': None} 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}}

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.

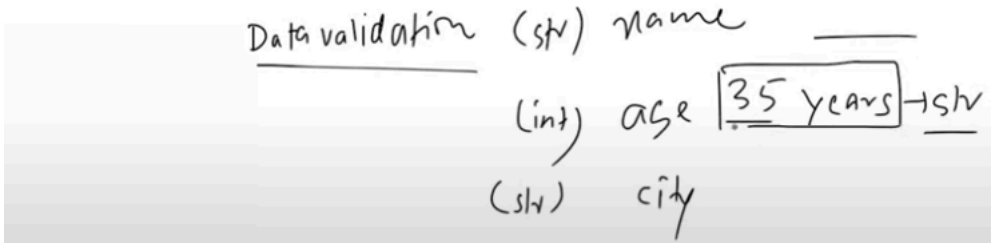
It works by defining a list of fields (ResponseSchema) that the model should return, ensuring the output follows a structured format.

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

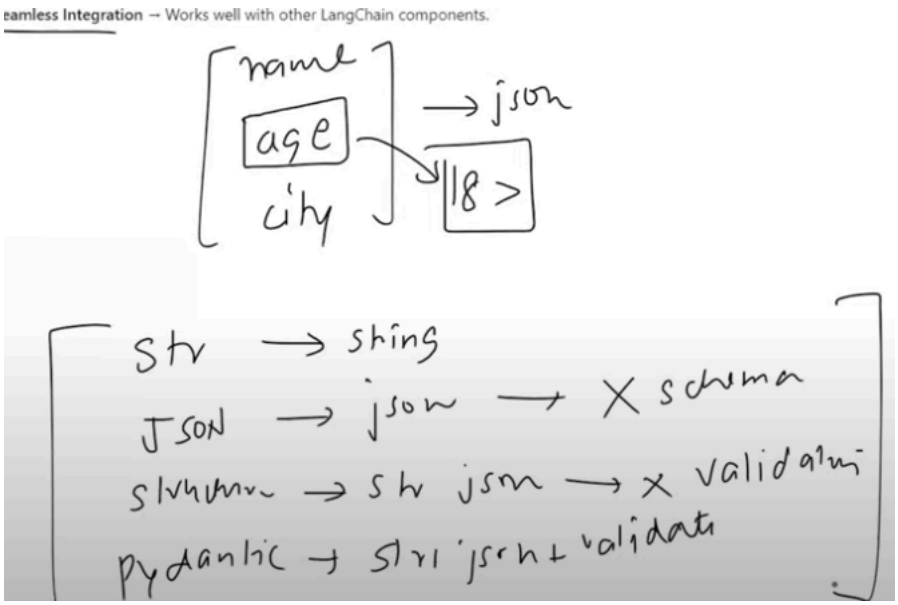
What is `PydanticOutputParser` in LangChain?

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

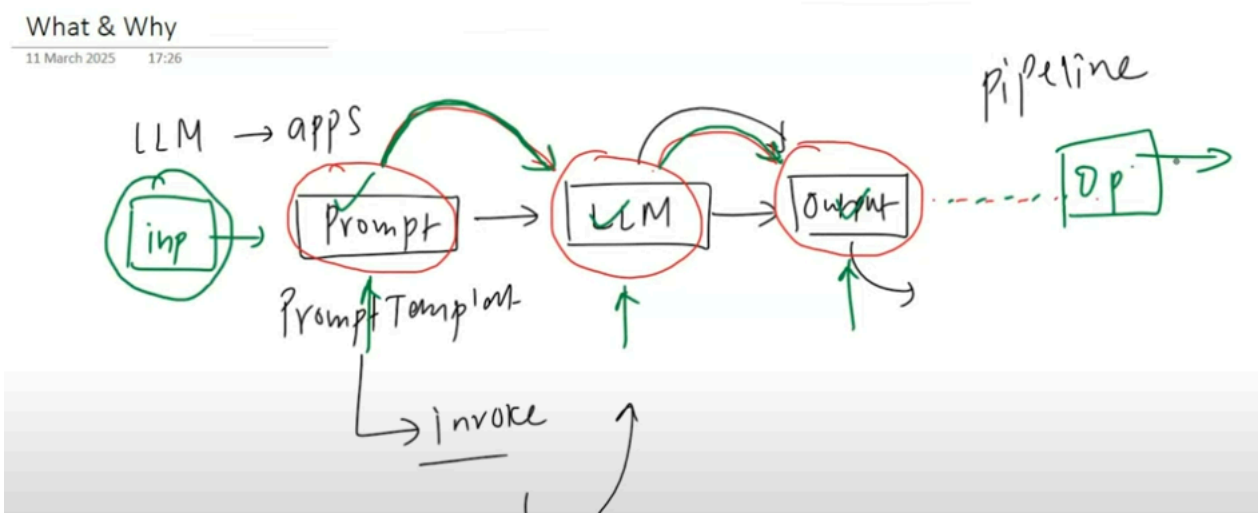
→ schema → [Pydantic Object]

Why Use `PydanticOutputParser`?

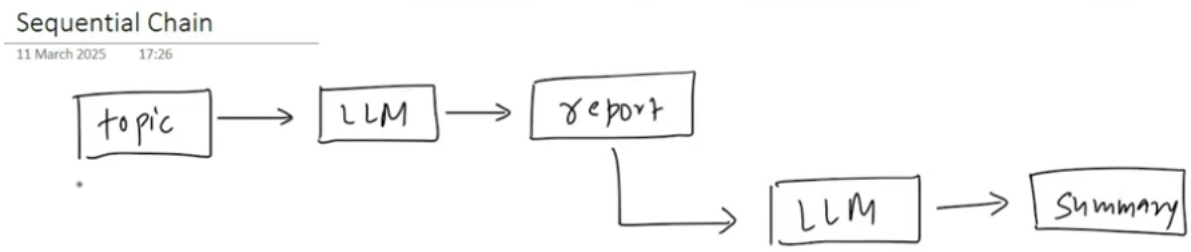
- **Strict Schema Enforcement** → Ensures that LLM responses follow a well-defined structure.
- **Type Safety** → Automatically converts LLM outputs into Python objects.
- **Easy Validation** → Uses Pydantic's built-in validation to catch incorrect or missing data.
- **Seamless Integration** → Works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



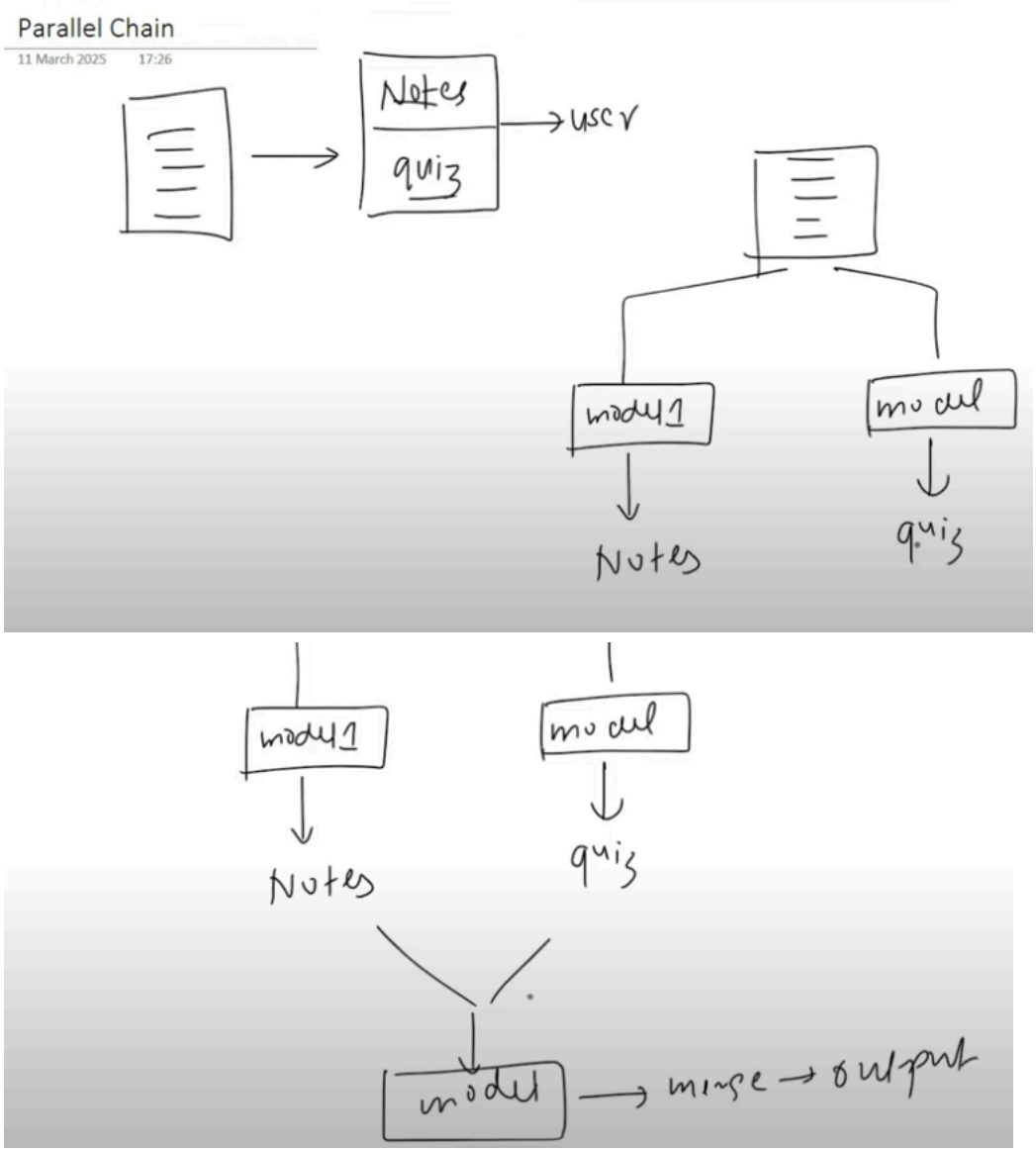
Chains

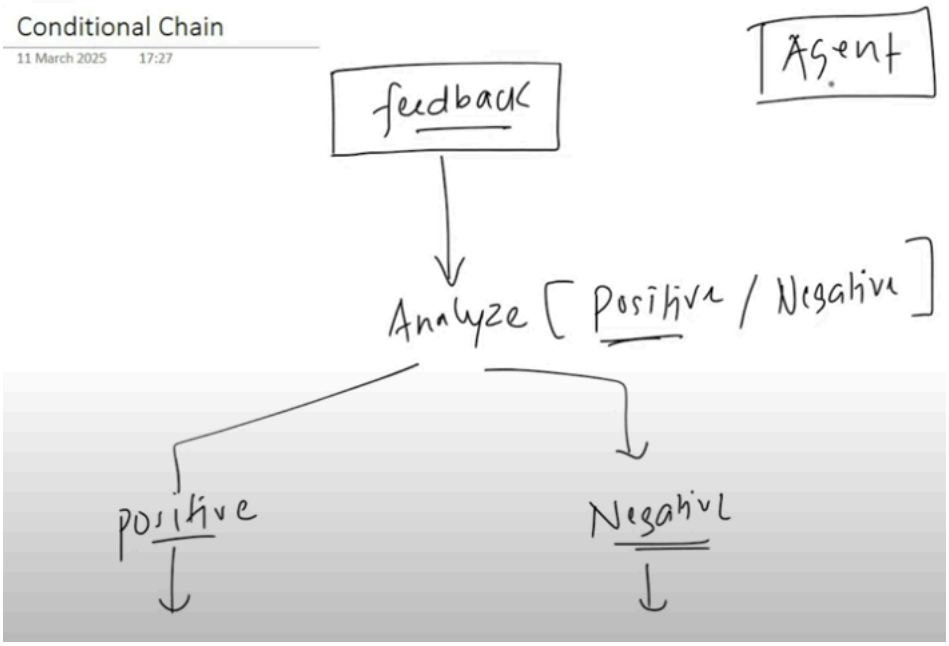


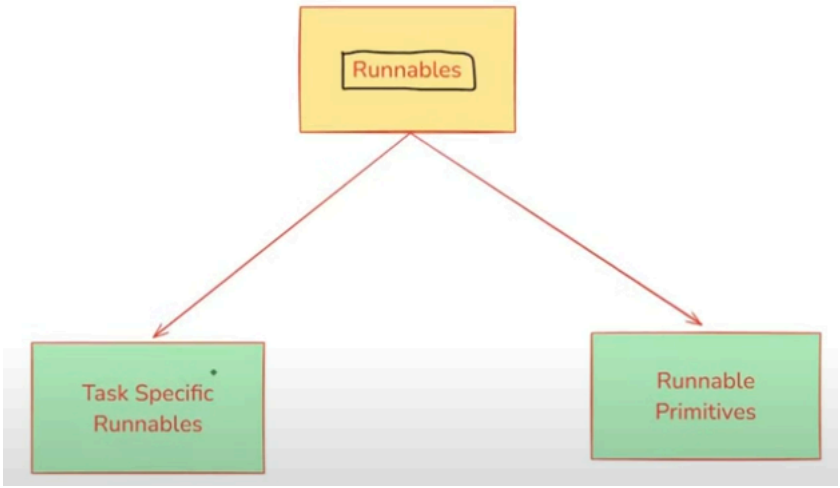

```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

load_dotenv()

prompt = PromptTemplate(
    template='Generate 5 interesting facts about {topic}',
    input_variables=['topic']
)

model = ChatOpenAI()

parser = StrOutputParser()

chain = prompt | model | parser

chain.invoke({'topic':'cricket'})
```

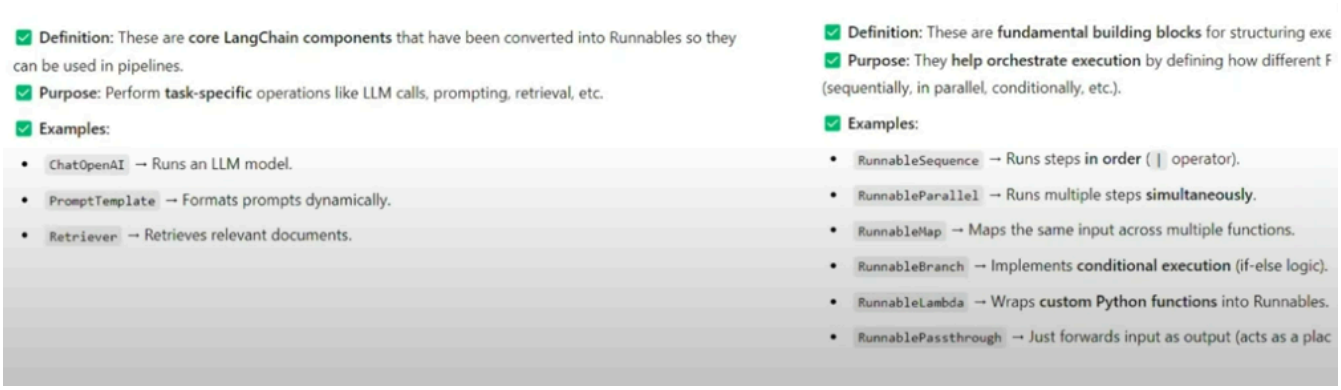


1. simple chain
2. sequential chain
3. parallel chain
4. conditional chain

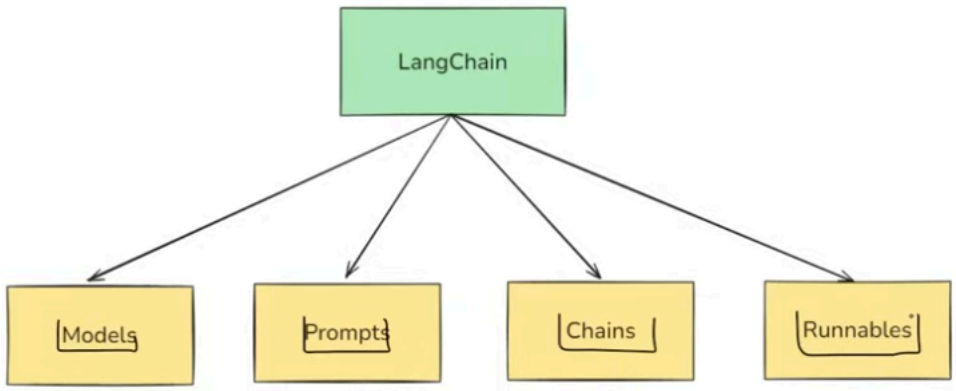


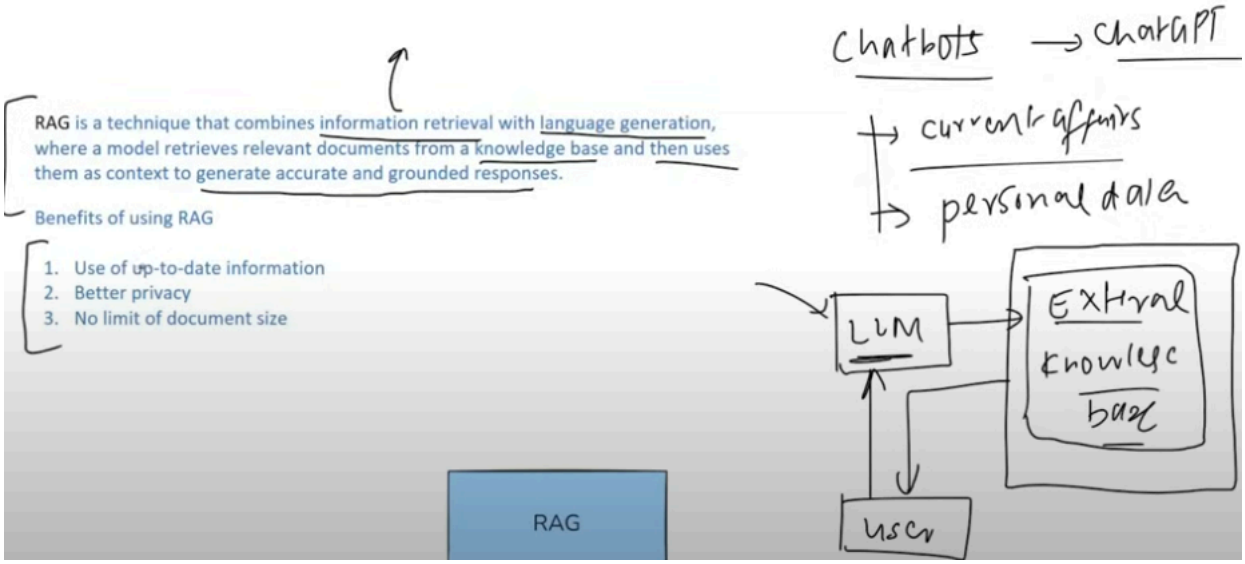


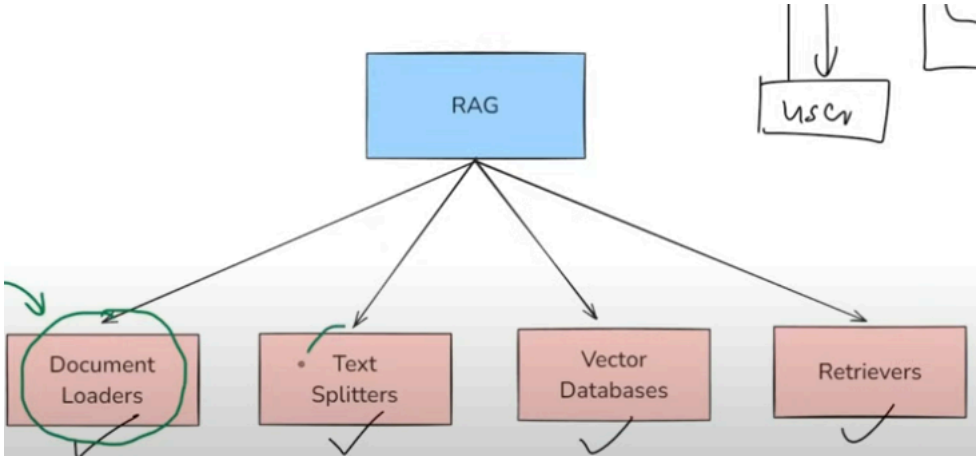


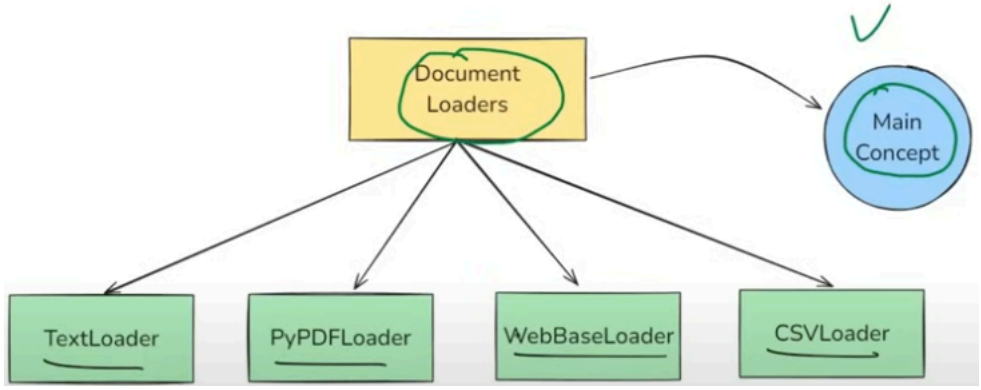


Skipped 2 videos on runnables


LangChain
Models
Prompts
Chains
Runnables


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
Chatbots → ChatGPT
current affairs
personal data
LLM
External knowledge base
User
RAG


RAG
User
Document Loaders
Text Splitters
Vector Databases
Retrievers

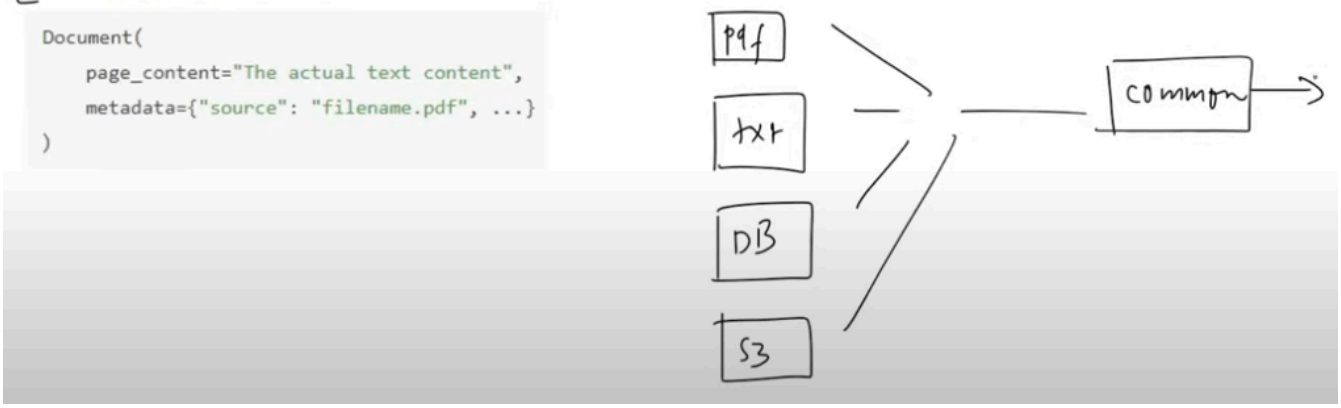

Document Loaders
Main Concept
TextLoader
PyPDFLoader
WebBaseLoader
CSVLoader

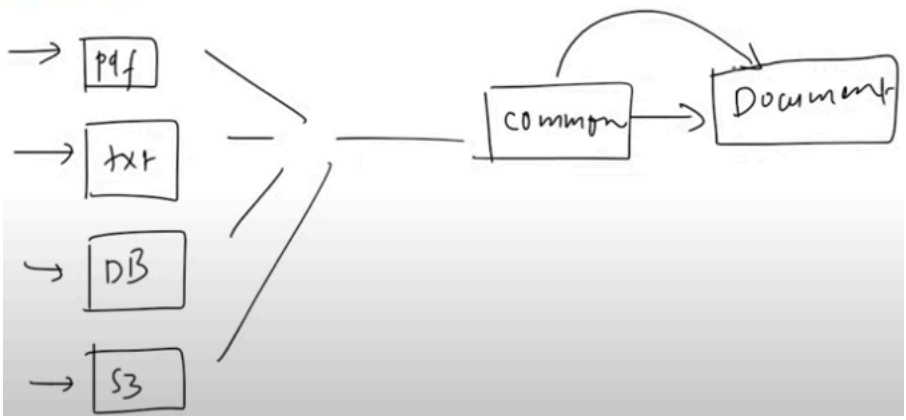
Document Loaders in LangChain

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

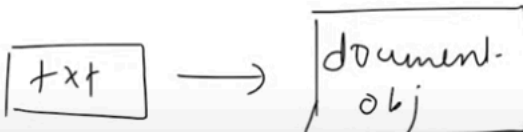
TextLoader is a simple and commonly used document loader in LangChain that reads plain text (.txt) files and converts them into LangChain Document objects.

Use Case

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

Limitation

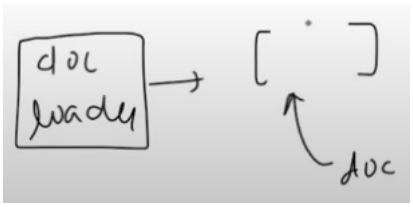
- Works only with .txt files



text_loader.py | cricket.txt

```python
from langchain_community.document_loaders import TextLoader

loader = TextLoader('cricket.txt', encoding='utf-8')

docs = loader.load()

print(type(docs))
```

```
ograms/Python/Python311/python.ex
t_loader.py
<class 'list'>
```



PyPDFLoader

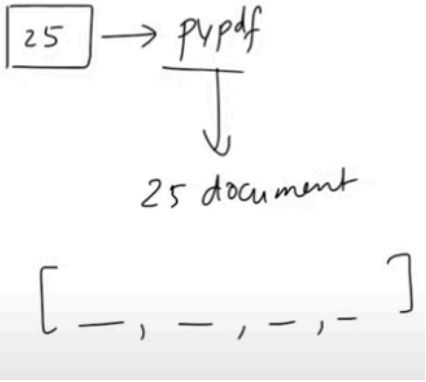
27 March 2025 17:50

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.



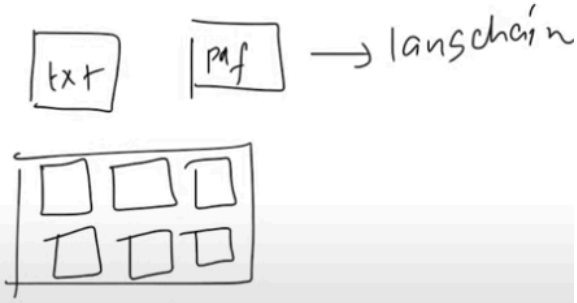
Refer: https://python.langchain.com/docs/integrations/document_loaders/

DirectoryLoader

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 →

Easier 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

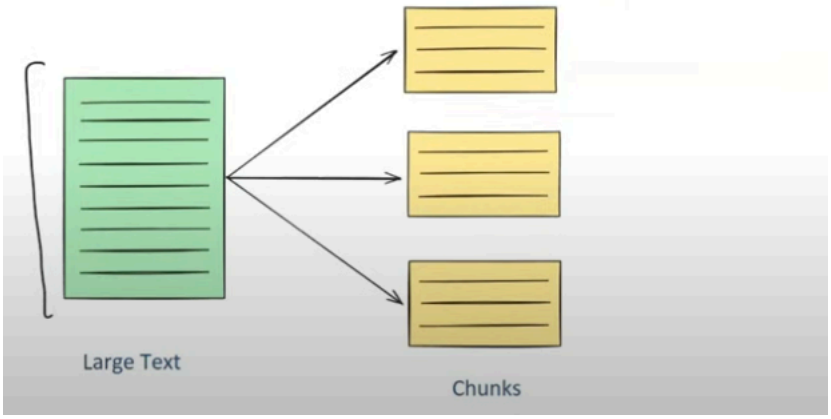
28 March 2025 01:48

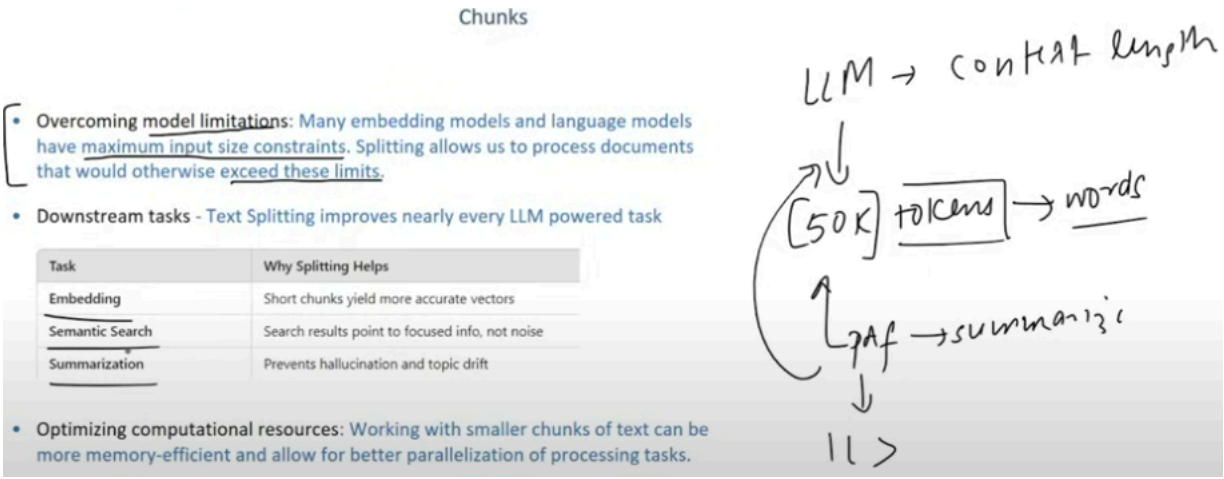
CSVLoader is a document loader used to load CSV files into LangChain Document objects — one per row, by default.

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.





✅ 1. Prevent Hallucination

Hallucination means:

> The model generates made-up, inaccurate, or misleading information not supported by the source data.

❗Why it happens during data splitting:

If a chunk:

- is **too short**, the model may lack enough context and fill in the blanks inaccurately.
- is **cut in the middle of a sentence or paragraph**, the meaning is lost, leading to guesses.

✅ How to prevent it:

- Use **semantic-aware splitters** like `RecursiveCharacterTextSplitter` that avoid breaking mid-sentence or mid-paragraph.
- Make sure each chunk is **complete and self-contained**, especially for factual queries.

✅ 2. Prevent Topic Drift

Topic Drift means:

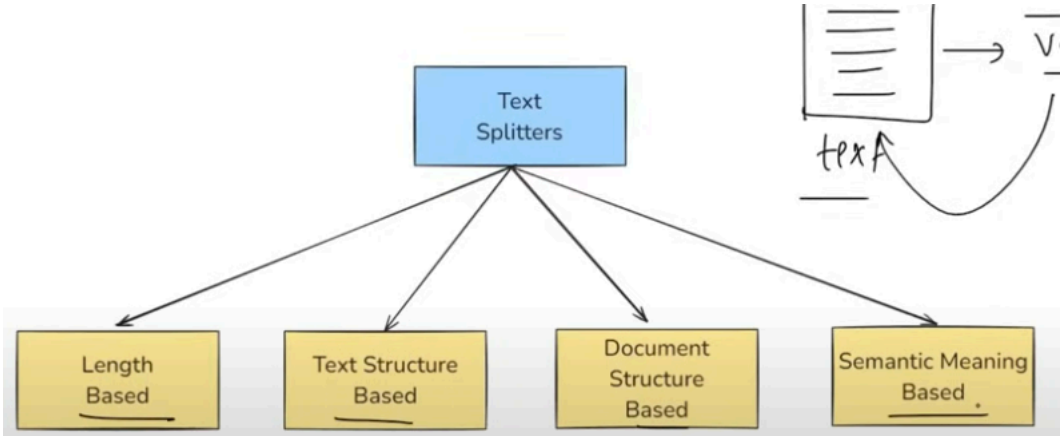
> When a chunk or passage contains **multiple unrelated topics**, it's harder for the retriever to match it with a specific user query.

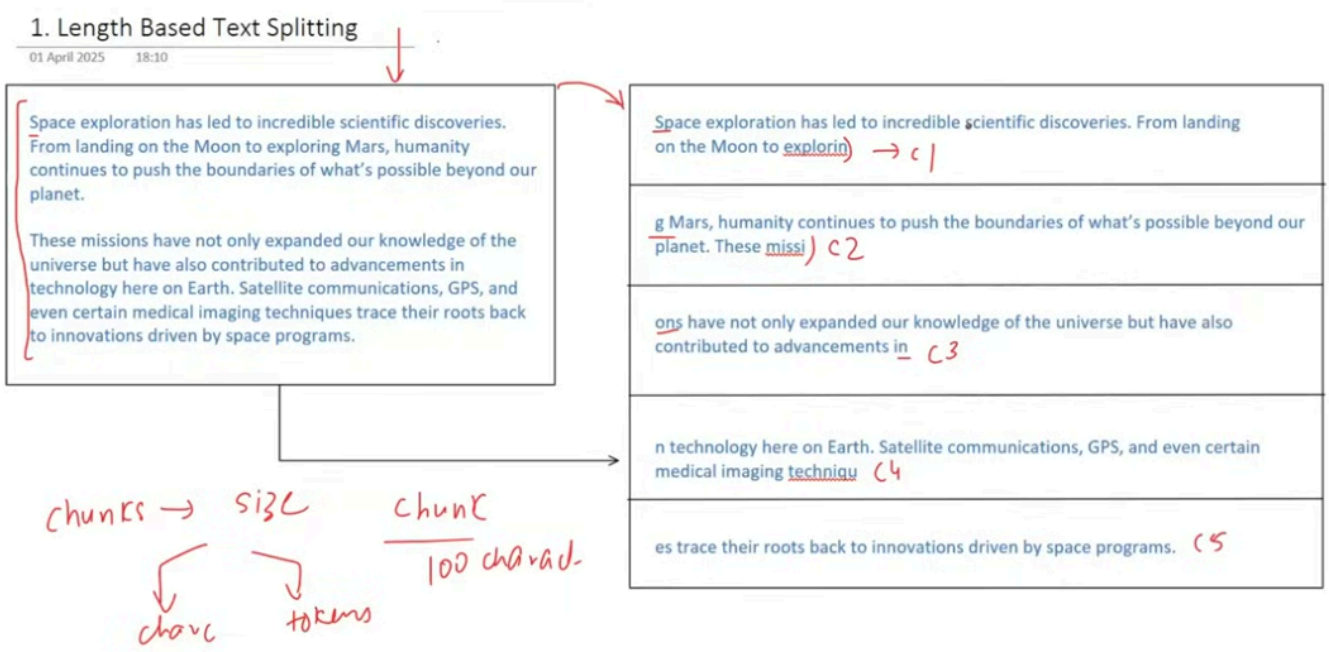
❗Why it happens during data splitting:

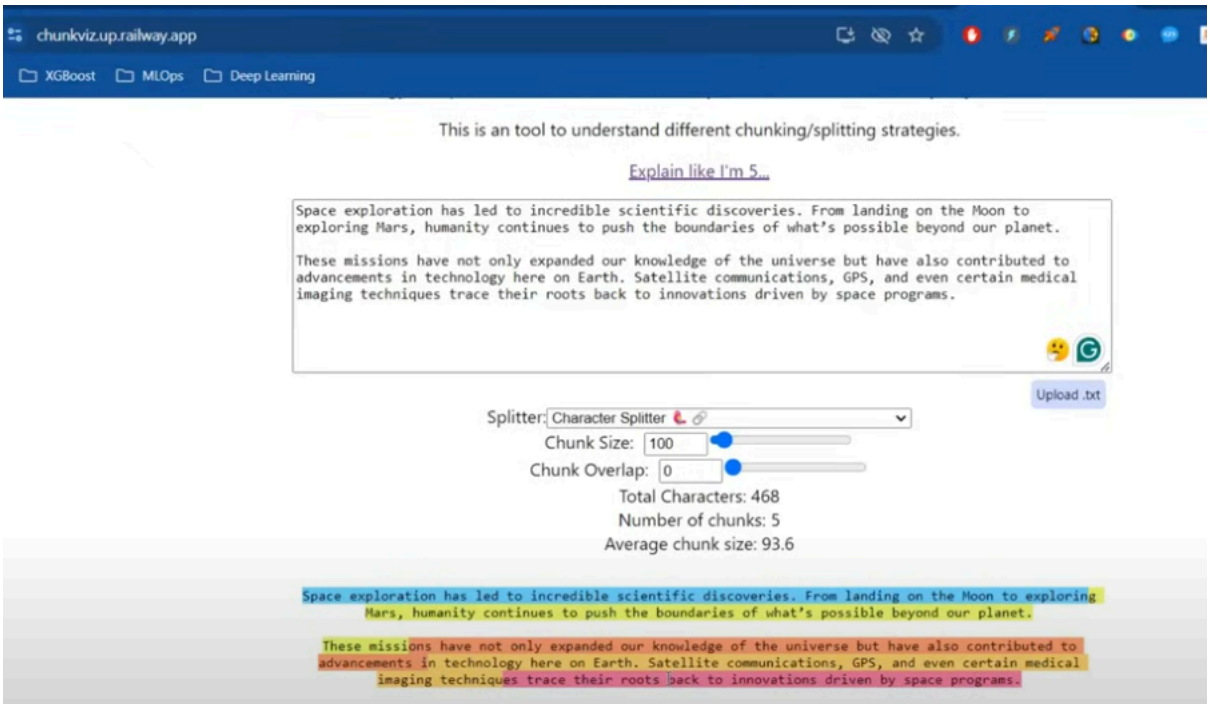
- If a chunk is **too large**, it may mix several topics or ideas.
- The retriever might fetch a chunk where only part of it is relevant — this distracts the model.

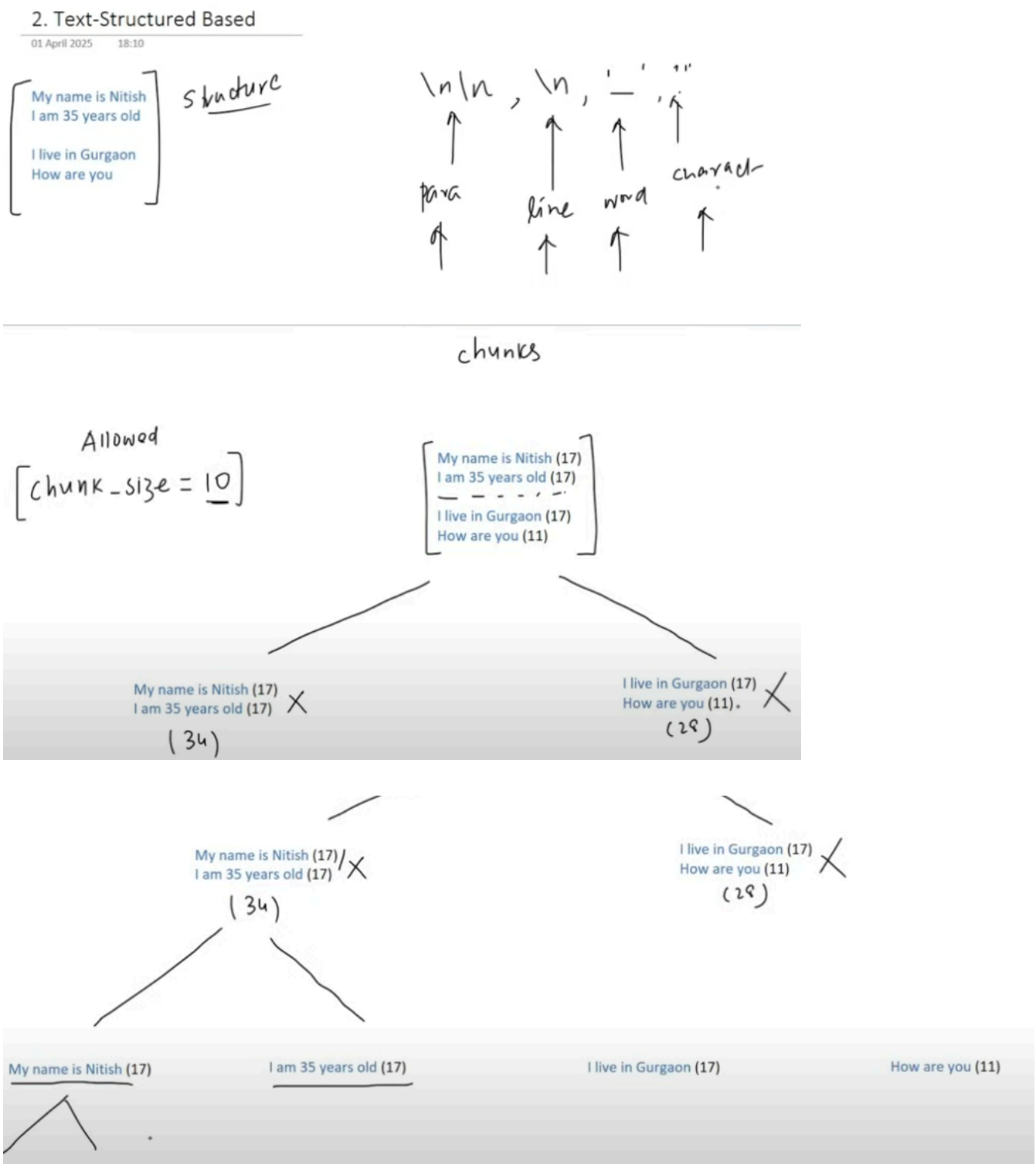
✅ How to prevent it:

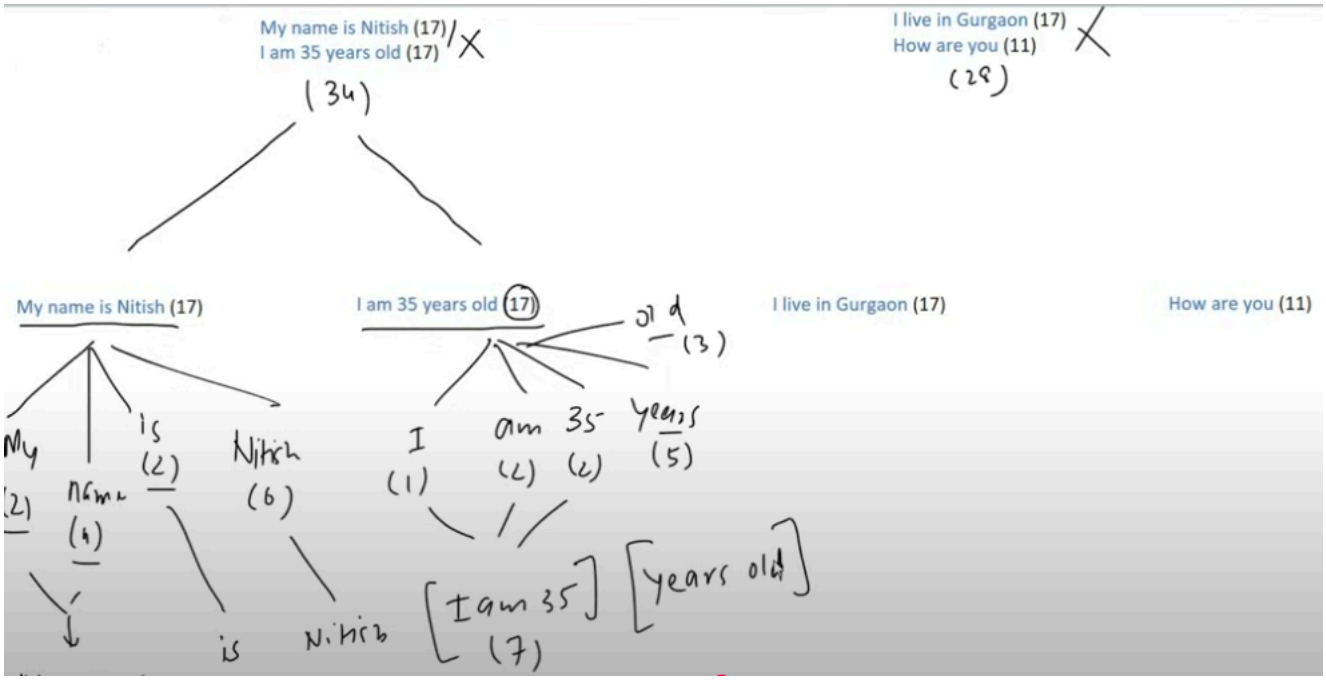
- Keep chunks **narrowly focused** on a single subject or paragraph.
- Choose a **chunk size and overlap** that respects topic boundaries.

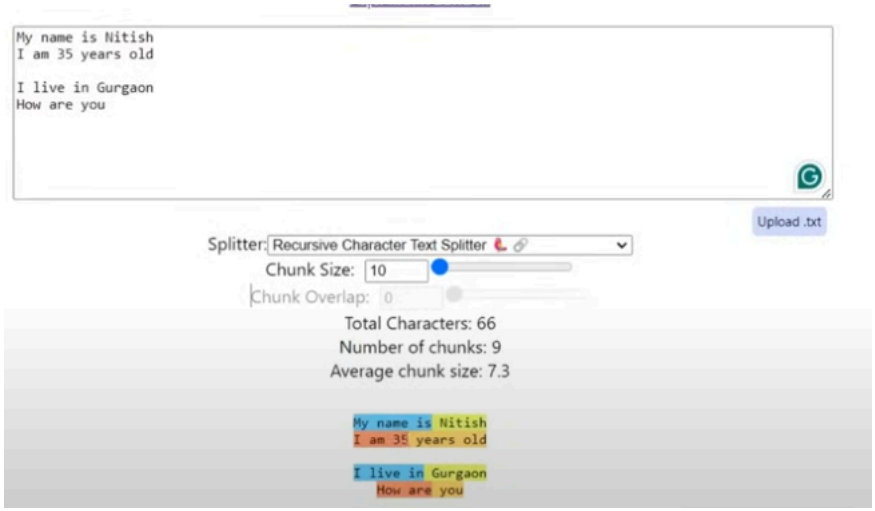

Text Splitters
Length Based
Text Structure Based
Document Structure Based
Semantic Meaning Based

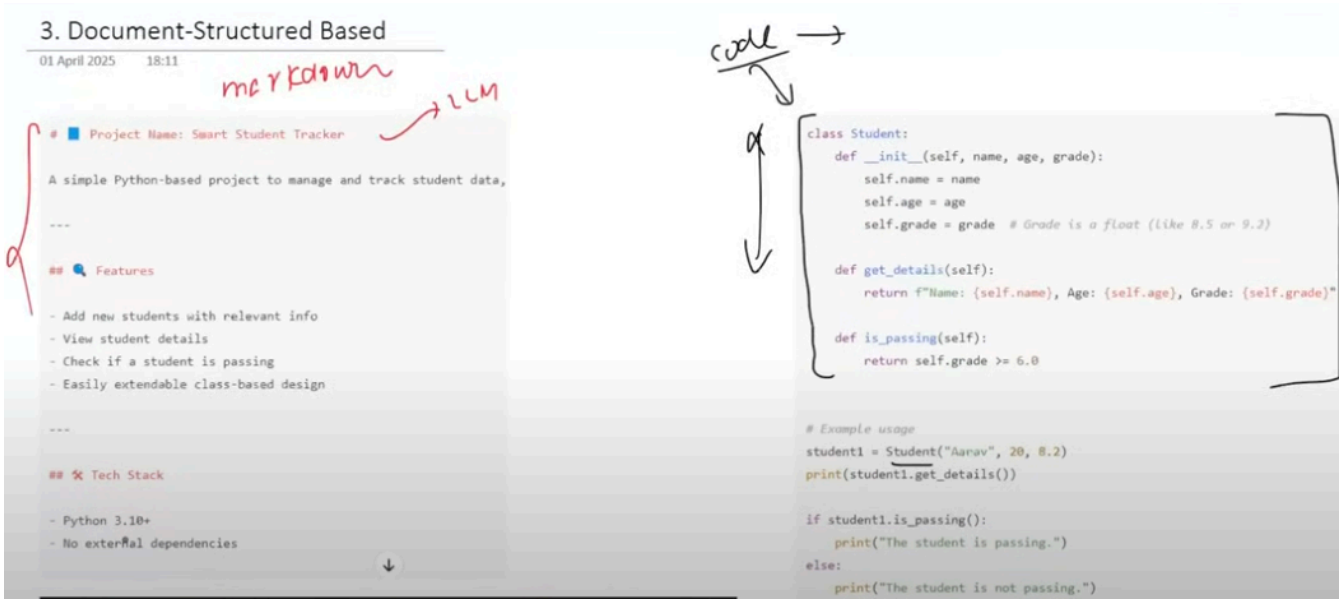

1. Length Based Text Splitting
01 April 2025 18:10
Space exploration has led to incredible scientific discoveries. From landing on the Moon to exploring Mars, humanity continues to push the boundaries of what's possible beyond our planet.
These missions have not only expanded our knowledge of the universe 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
g Mars, humanity continues to push the boundaries of what's possible beyond our planet. These missi
ons have not only expanded our knowledge of the universe but have also contributed to advancements in
n technology here on Earth. Satellite communications, GPS, and even certain medical imaging techniqu
es trace their roots back to innovations driven by space programs.


chunkviz.up.railway.app
XGBoost
MLOps
Deep Learning
This is an tool to understand different chunking/splitting strategies.
Explain like I'm 5...
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.
Upload .txt
Splitter: Character Splitter
Chunk Size: 100
Chunk Overlap: 0
Total Characters: 468
Number of chunks: 5
Average chunk size: 93.6
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.


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
\n\n , \n , ' ' ' '
para
line
word
charact-
chunks
Allowed
chunk_size = 10
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)
I am 35 years old (17)
(34)
I live in Gurgaon (17)
How are you (11).
(29)
My name is Nitish (17)
I am 35 years old (17)
(34)
I live in Gurgaon (17)
How are you (11)
(29)
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)
I am 35 years old (17)
(34)
I live in Gurgaon (17)
How are you (11)
(28)
My name is Nitish (17)
I am 35 years old (17)
I live in Gurgaon (17)
How are you (11)
[I am 35]
(7)
[years old]


My name is Nitish
I am 35 years old
I live in Gurgaon
How are you
Upload .txt
Splitter: Recursive Character Text Splitter
Chunk Size: 10
Chunk Overlap: 0
Total Characters: 66
Number of chunks: 9
Average chunk size: 7.3


3. Document-Structured Based
01 April 2025 18:11
markdown
LLM
code
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
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",
" ",
"",
```

```
print("The student is not passing.")
```

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

Splitter: Recursive Character Text Splitter - Python

Chunk Size: 350

Chunk Overlap: 0

Total Characters: 524

Number of chunks: 2

Average chunk size: 262.0

```
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.")
```

python_code_splitting.py × markdown_splitting.py

python_code_splitting.py > ...

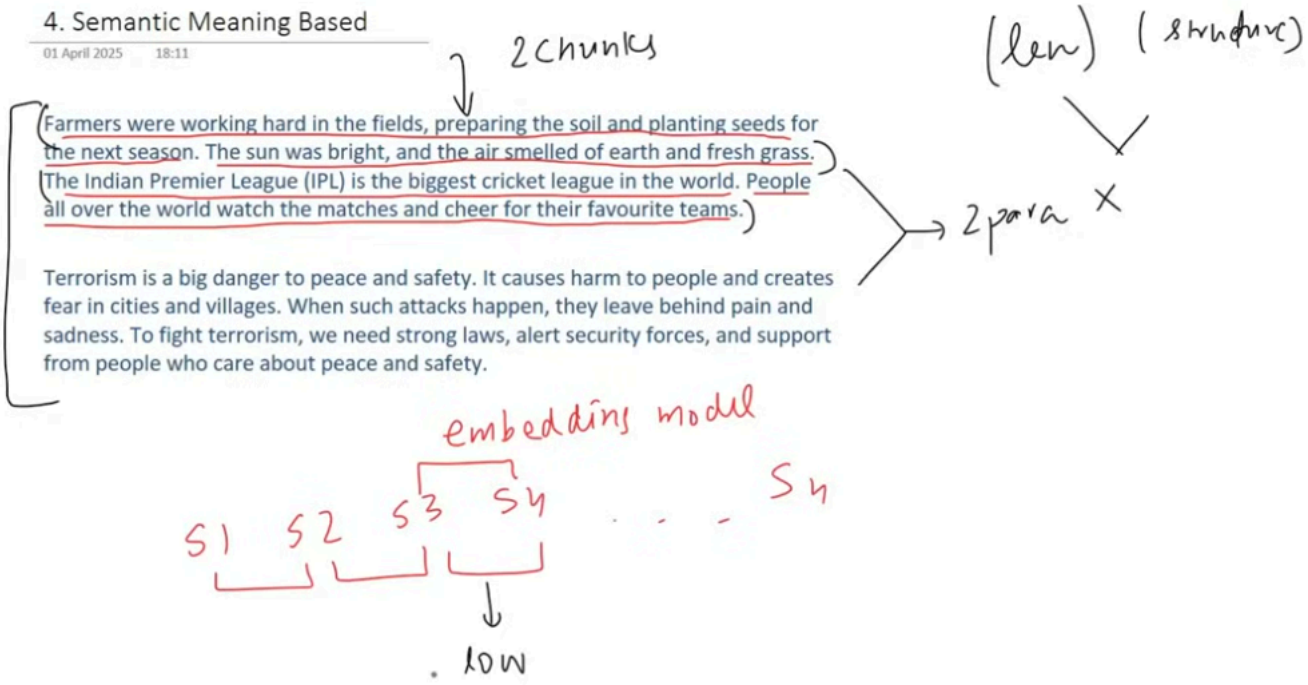
```
23  else:
24      print("The student is not passing.")
25
26  """
27
28  # Initialize the splitter
29  splitter = RecursiveCharacterTextSplitter.from_language(
30      language=Language.PYTHON,
31      chunk_size=300,
32      chunk_overlap=0,
33  )
34
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

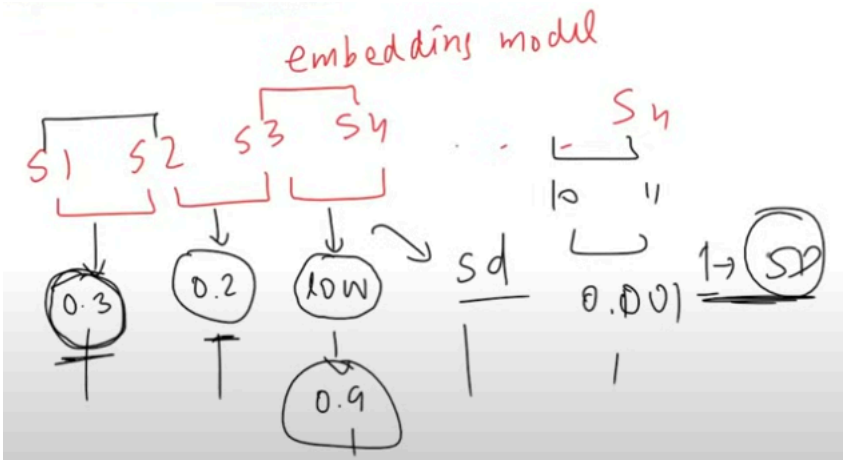
```
PS C:\Users\Nitish\Desktop\langchain-text-splitters>
```

4. Semantic Meaning Based

01 April 2025 18:11



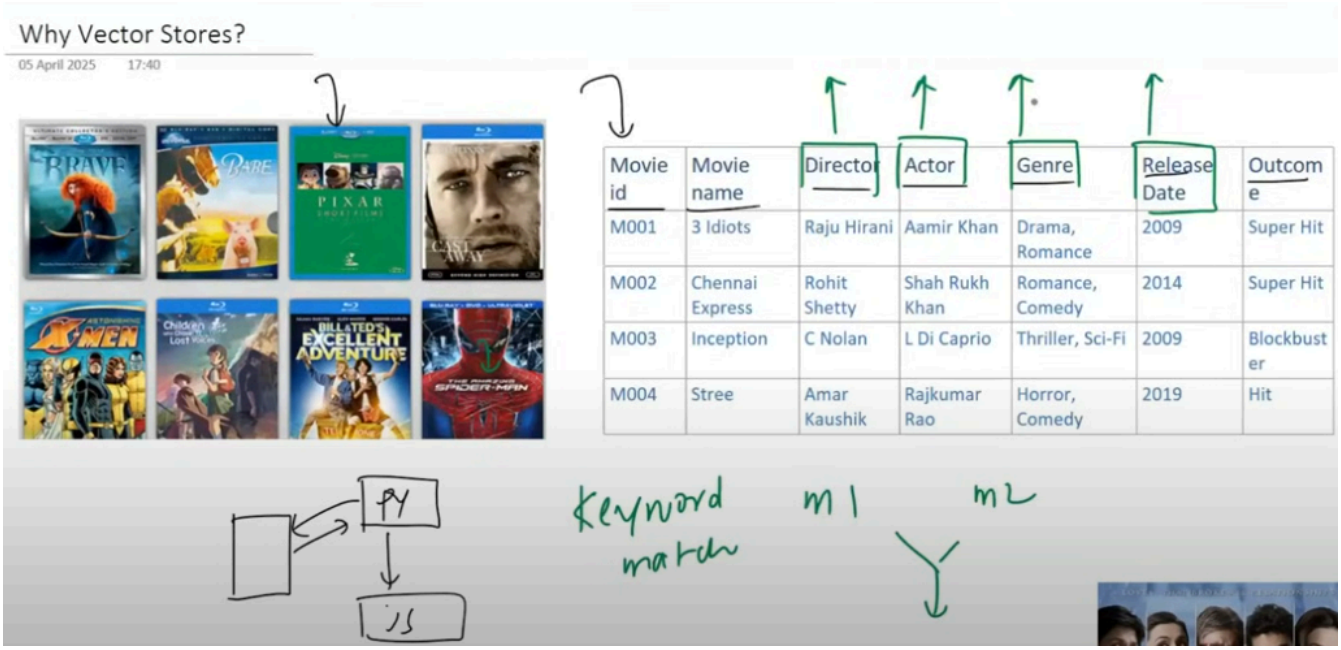

```python
# python_code_splitting.py | markdown_splitting.py | semantic_meaning_based.py
from langchain_experimental.text_splitter import SemanticChunker
from langchain_openai.embeddings import OpenAIEmbeddings
from dotenv import load_dotenv

load_dotenv()

text_splitter = SemanticChunker(
    OpenAIEmbeddings(), breakpoint_threshold_type="standard_deviation",
    breakpoint_threshold_amount=1
)

sample = """
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
```

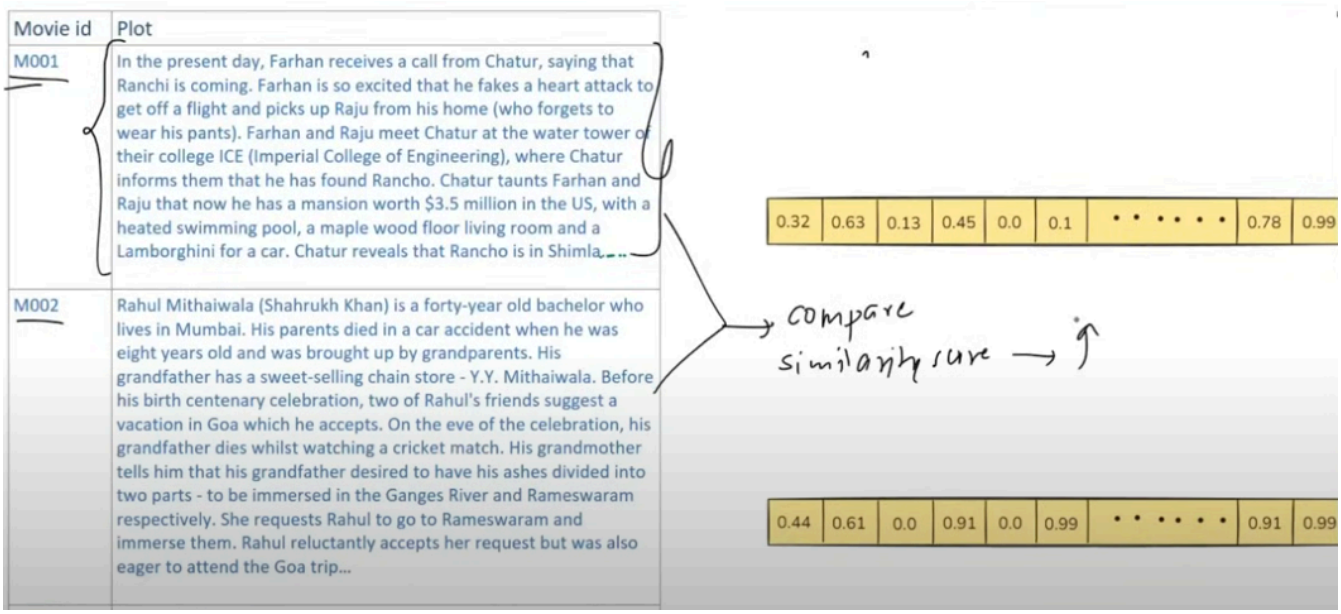


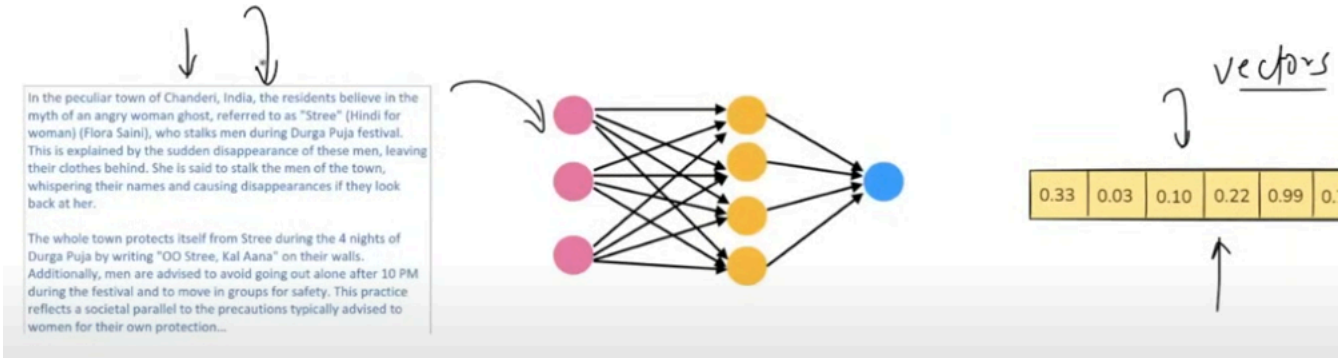
🧮 How Does the Standard Deviation Method Work?

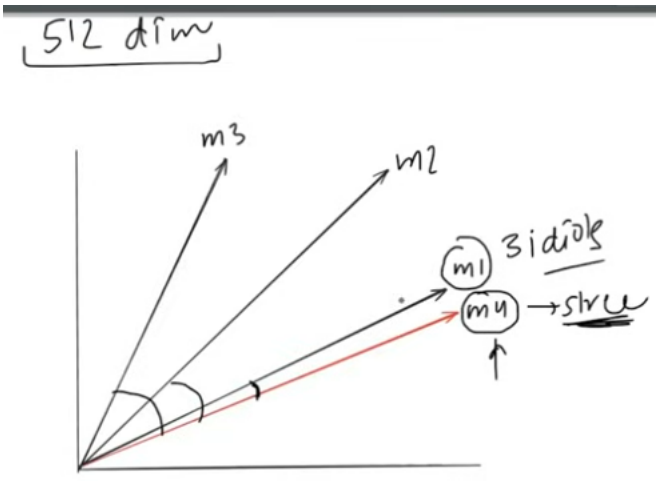
Step-by-Step:

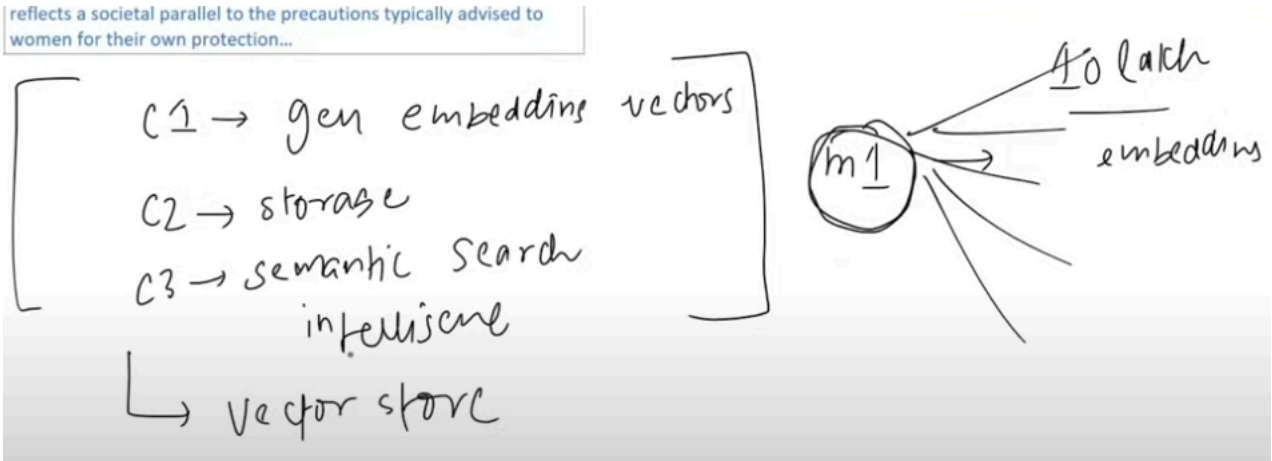
1. **Embed each sentence or paragraph** using an embedding model (e.g., OpenAI, HuggingFace).
2. **Compute pairwise similarity** between consecutive embeddings (typically cosine similarity).
3. **Calculate the mean and standard deviation (σ)** of these similarity scores.
4. **Identify low-similarity points** where the similarity drops significantly — specifically where it drops **below** `mean - σ` or `mean - n*σ` (n = multiplier, e.g., 1 or 1.5).
5. **Split at those low-similarity points**, because they likely represent topic or context shifts.




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...
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


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...
vectors
0.33 0.03 0.10 0.22 0.99

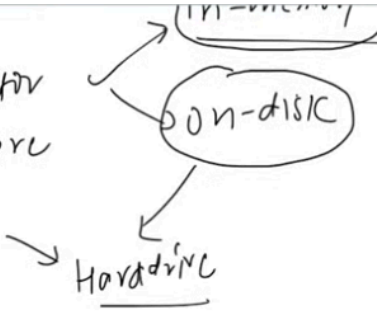

512 dim
m3
m2
m1
m4


reflects a societal parallel to the precautions typically advised to women for their own protection...
C1 → gen embedding vectors
C2 → storage
C3 → semantic search
intelligence
↳ vector store
m1
40 lakh
embeddings

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

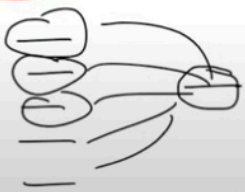
Key Features

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.

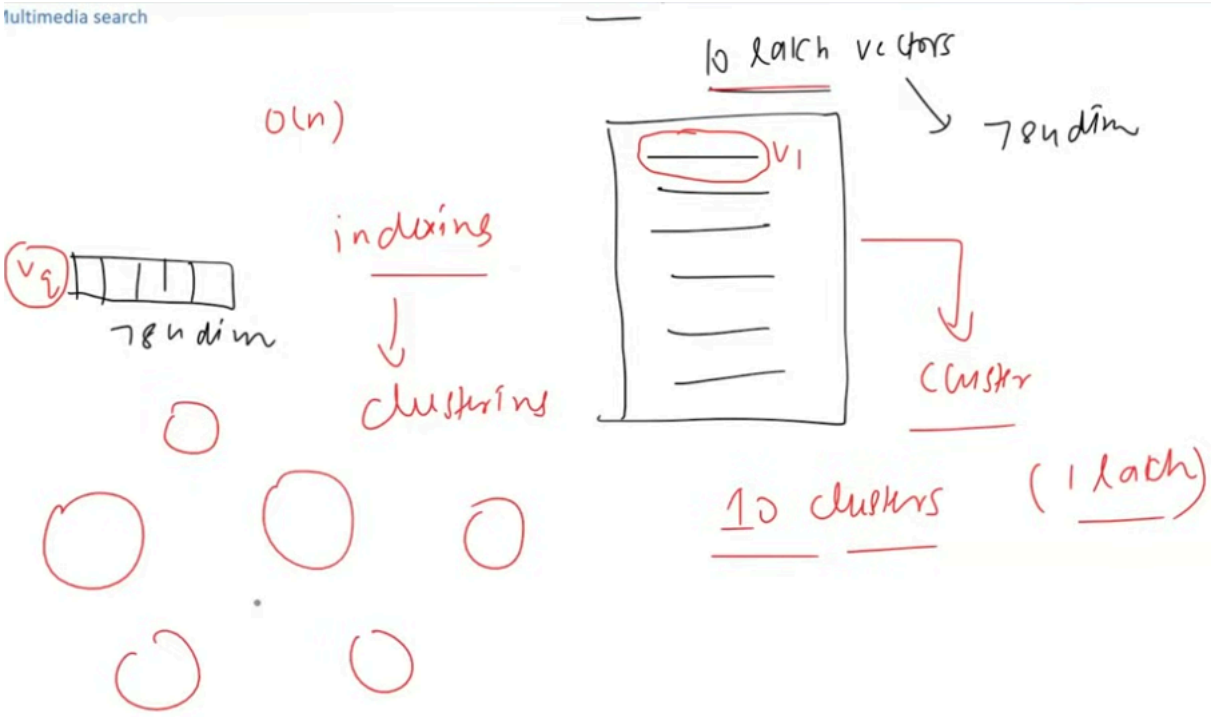


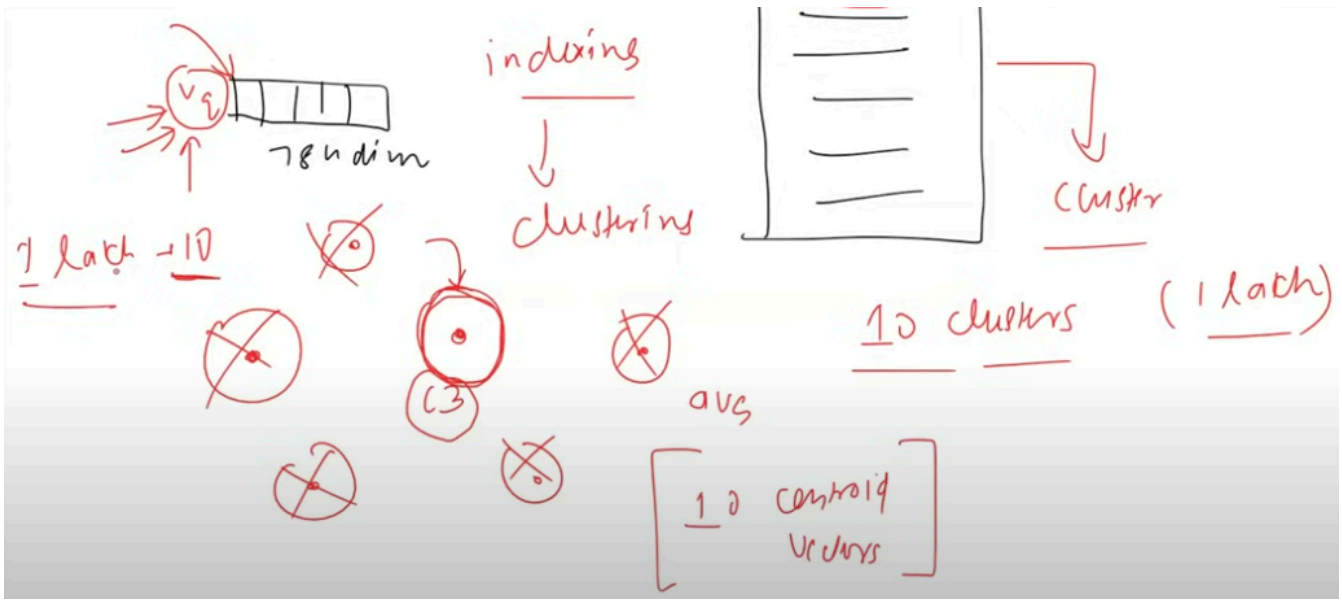
Use-cases

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



Multimedia search

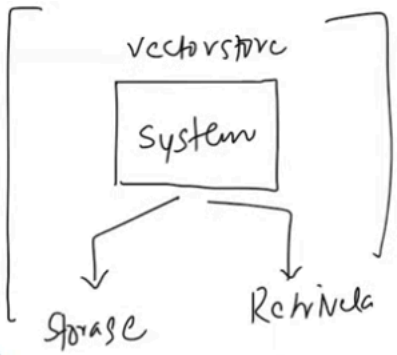




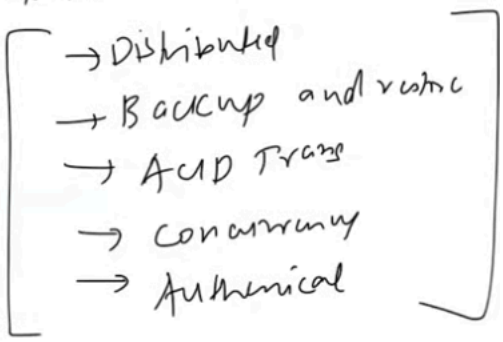
Vector Store Vs Vector Database

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



- 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

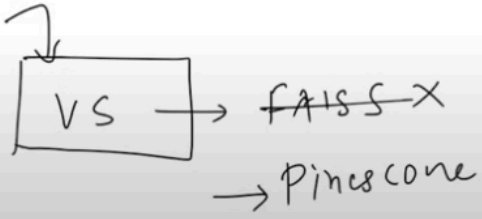
- Supported Stores: LangChain integrates with multiple vector stores (FAISS, Pinecone, Chroma, Qdrant, Weaviate, etc.), giving you flexibility in scale, features, and deployment.
- Common Interface: A uniform Vector Store API lets you swap out one backend (e.g., FAISS) for another (e.g., Pinecone) with 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.

`from_documents(...)` or `from_texts(...)`

`add_documents(...)` or `add_texts(...)`

`similarity_search(query, k=...)`

Metadata-Based Filtering



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.