What happens in chat gpt?

- You ask a question you get ans immediately
- But as chatgpt is trained on past data, when you ask chatgpt about **current affairs, personal information like about ur personal email**
- **It wont answer you**

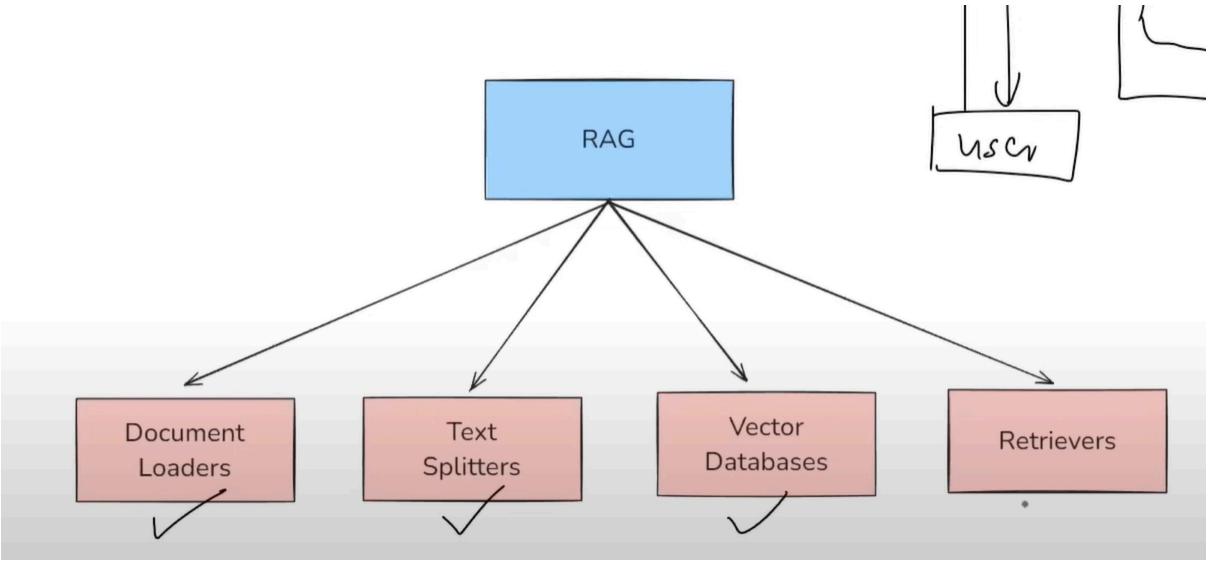
So in this situation you use RAG (which is connected to external database)
So what happens when a user asks a question to llm which the **llm can't answer** now with the help of rag the llm can take help from an external database.

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.

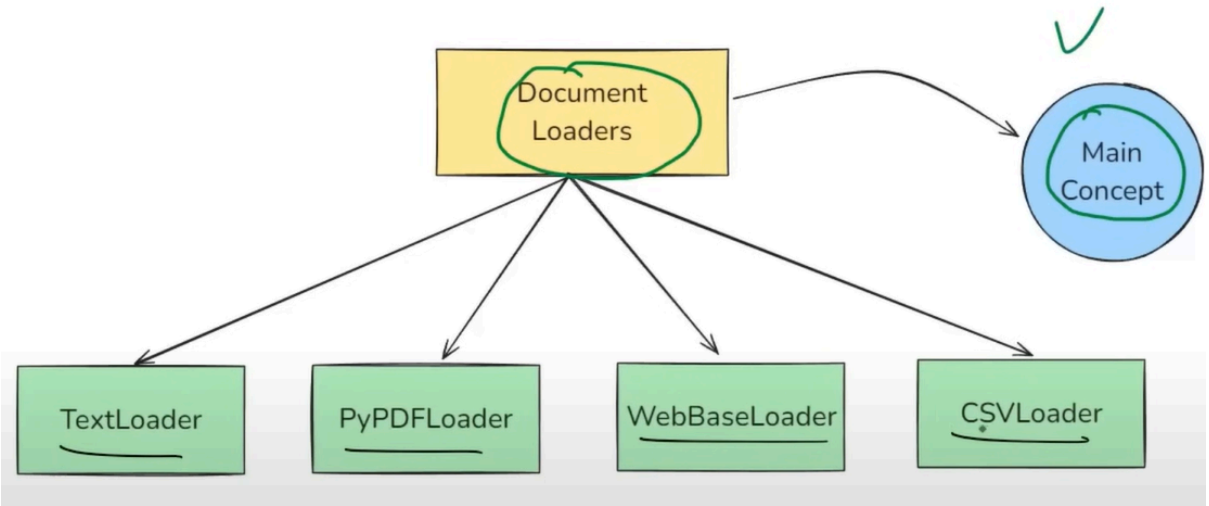
Benefits of using RAG

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

- Privacy : U want to ask questions on personal data u cant upload confidential info on chatgpt
- No limit of document size: Chatgpt has limited context length like 1 gb document. U can upload on rag which divides the document in chunks



Document Loaders



Document loaders : 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"}

)

1. TextLoader

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

txt → document-obj

2. PyPdfLoader

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"}),
    ...
]
```

25 → PyPdf

↓

25 document

[_ , _ , _ , _]

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`

3. DirectoryLoader

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()`

- **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.

directory_loader.py > ...

```python
from langchain_community.document_loaders import DirectoryLoader, PyPDFLoader

loader = DirectoryLoader(
    path='books',
    glob='*.pdf',
    loader_cls=PyPDFLoader
)

docs = loader.load()

print(docs[325].page_content)
```

So what is happening in lazy_load one document is entering in memory writing its metadata and then deleting from memory

text_loader.py | pdf_loader.py | directory_loader.py

directory_loader.py > ...

```python
    path='books',
    glob='*.pdf',
    loader_cls=PyPDFLoader
)

docs = loader.lazy_load()

for document in docs:
    print(document.metadata)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
{'producer': 'Adobe PDF Library 10.0.1', 'creator': 'Adobe InDesign CS6 (Windows)', 'crea
tiondate': '2016-01-27T18:55:12+05:30', 'moddate': '2016-01-30T17:30:05+05:30', 'trapped'
: '/False', 'source': 'books\\Practical Machine Learning.pdf', 'total_pages': 468, 'page'
: 465, 'page_label': '435'}
{'producer': 'Adobe PDF Library 10.0.1', 'creator': 'Adobe InDesign CS6 (Windows)', 'crea
tiondate': '2016-01-27T18:55:12+05:30', 'moddate': '2016-01-30T17:30:05+05:30', 'trapped'
: '/False', 'source': 'books\\Practical Machine Learning.pdf', 'total_pages': 468, 'page'
: 466, 'page_label': '436'}
{'producer': 'Adobe PDF Library 10.0.1', 'creator': 'Adobe InDesign CS6 (Windows)', 'crea
tiondate': '2016-01-27T18:55:12+05:30', 'moddate': '2016-01-30T17:30:05+05:30', 'trapped'
```

4. WebBaseLoader

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).

Uses 2 library :

1. request library: make http request to that web page
2. Beautifulsoup: parse the html context to extract text

5. CSVLoader

- Here every row acts as Document if there are 256 rows there are 256 documents in total

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.