RAG & Agents: Architecting Knowledge-Augmented Autonomy

Palacode Narayana Iyer Anantharaman

1st Dec 2025

Unlocking Enterprise Intelligence: Why We Need Retrieval

❗ **Limitations of LLMs:**

Parametric knowledge is static — can't answer questions beyond training data.

No awareness of private or proprietary information (e.g. internal specs, designs, source code).

Context window is bounded — even with 1M tokens, not enough for many real-world corpora.

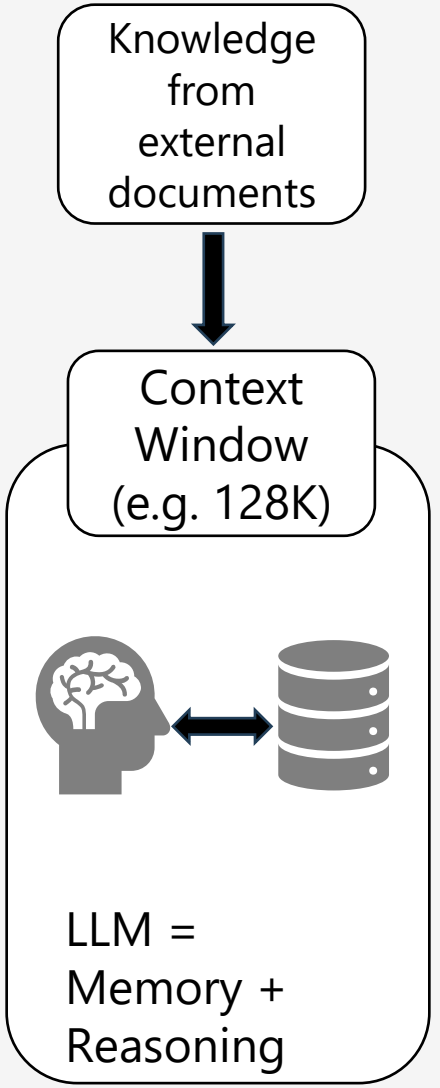
Trained on data with a cutoff — lacks recent developments or live updates.

🚀 **Enter RAG (Retrieval-Augmented Generation):**

Allows **dynamic retrieval** from external or private sources.

Enables **Q&A, summarization, and reasoning** on top of **enterprise documents**.

Helps optimize the **use of limited context window** by retrieving only the relevant chunks

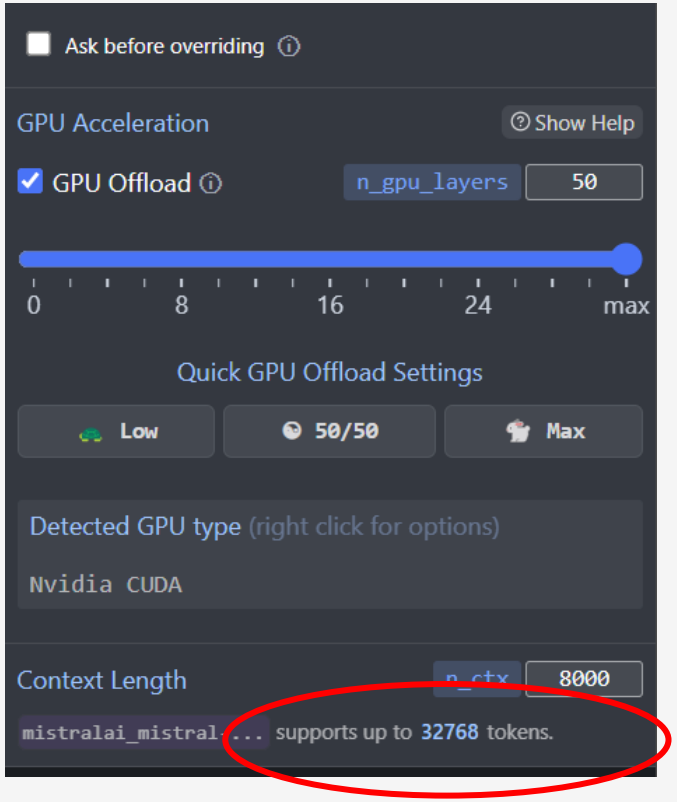


Context Limitations

LLM's are unaware of concepts outside of their training set

Filling gaps in knowledge with assumptions

Very hard to teach LLM's about new concepts



Euclidean Distance and Cosine Similarity

Euclidean Distance is defined as:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Where x and y are two vectors. Or:

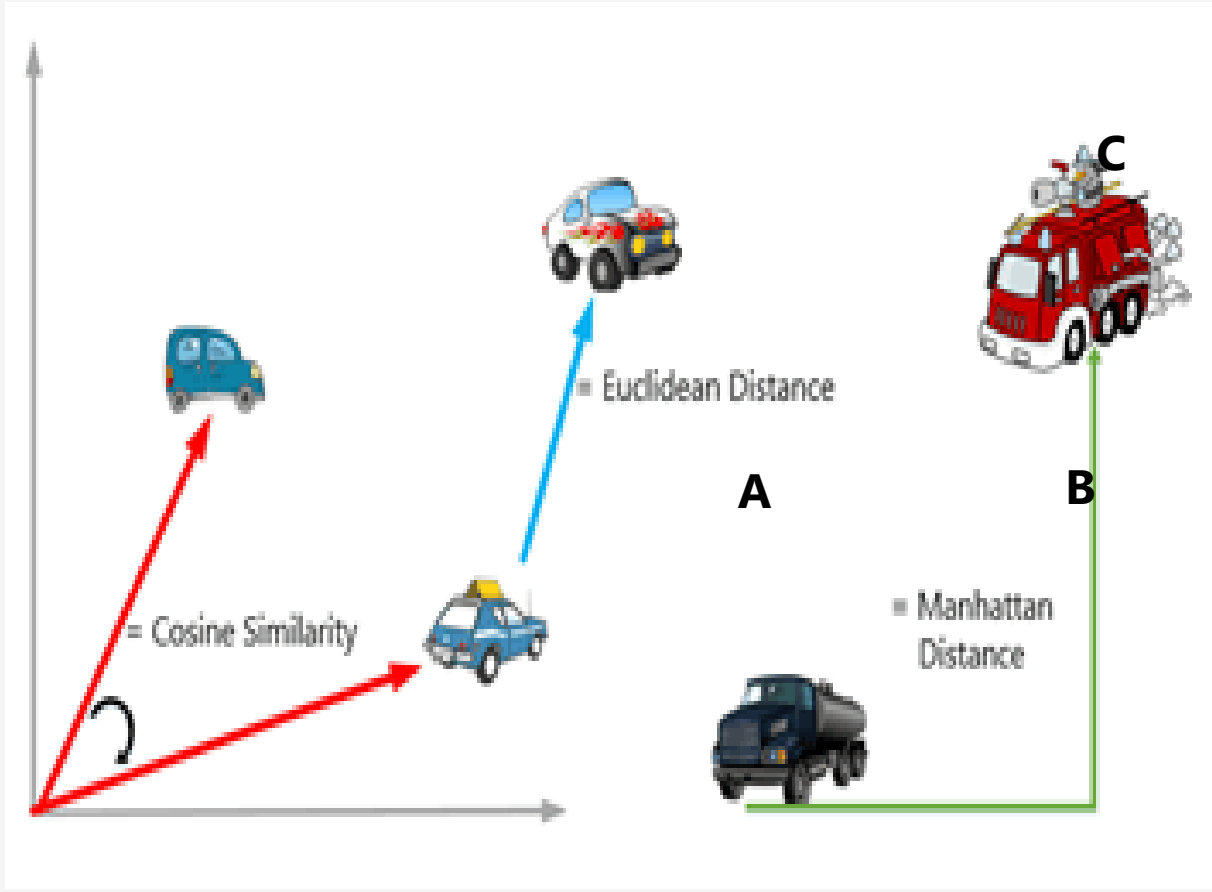
```
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y) ** 2))
```

Cosine Similarity:

$$\frac{x \bullet y}{\sqrt{x \bullet x}\sqrt{y \bullet y}}$$

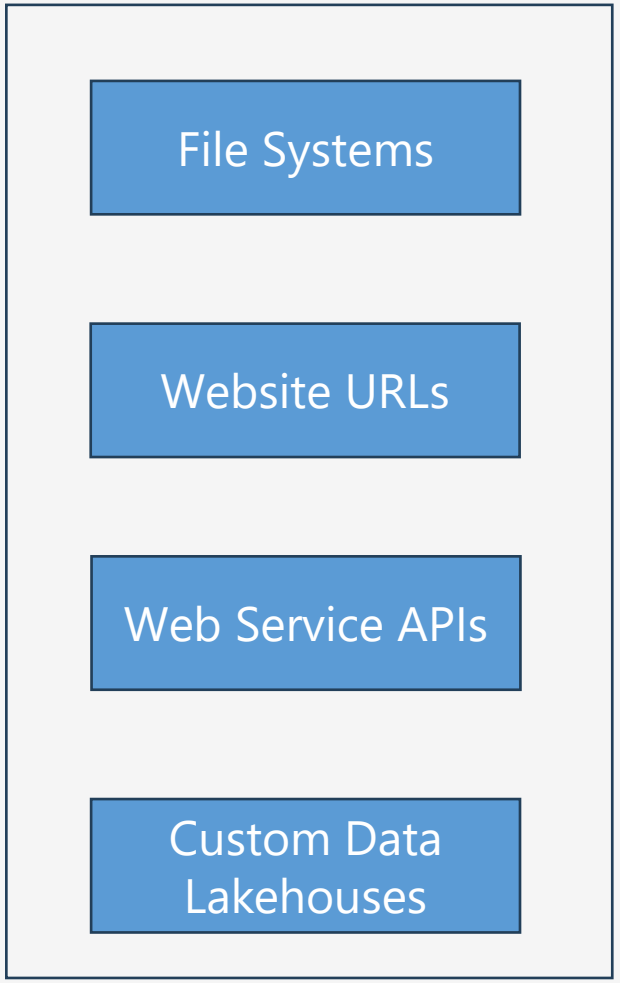
Where x and y are two vectors. Or:

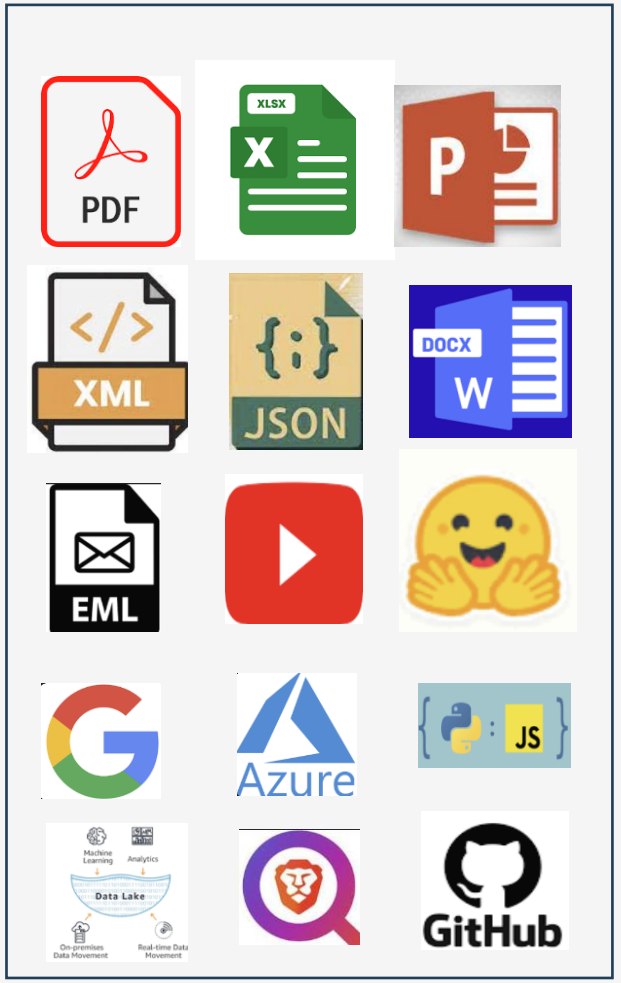
```
def cosine_similarity(x, y):
    return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))
```



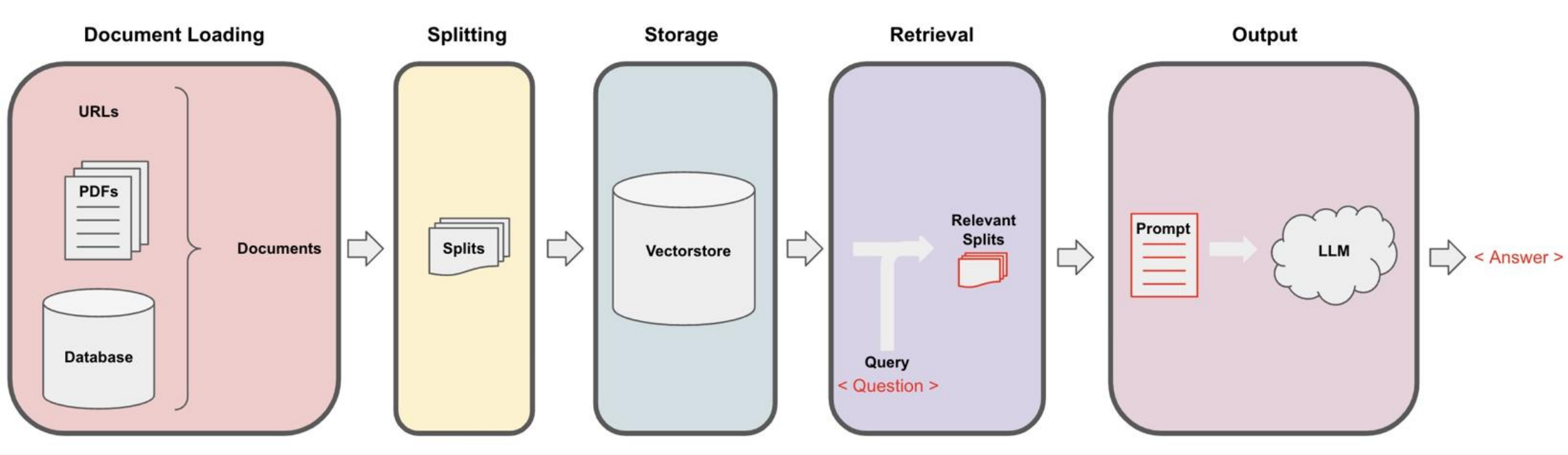
RAG Big Picture: Data Sources and Data formats

- Data comes from diverse sources: local files, cloud storage, APIs, SaaS tools, emails, etc.
- Includes both unstructured (PDFs, text) and structured (JSON, Excel) data — plus images, videos, code, and more.
- File formats vary: .pdf, .xlsx, .pptx, .json, etc.
- LangChain doc loaders standardize access using a common Document interface.





LangChain Pipeline for Question Answering

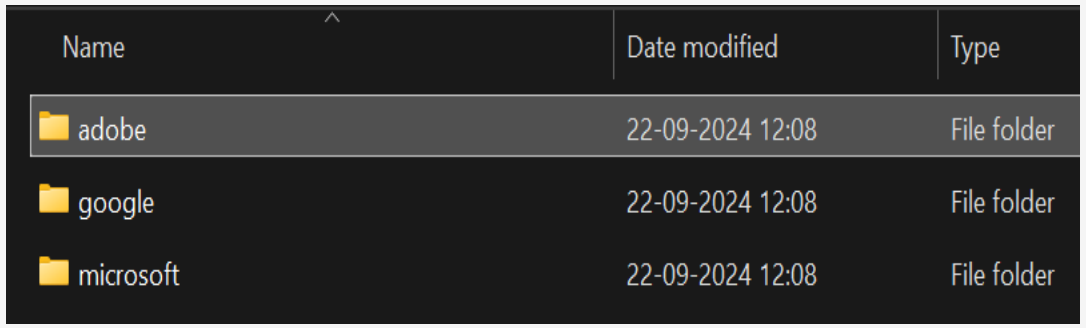


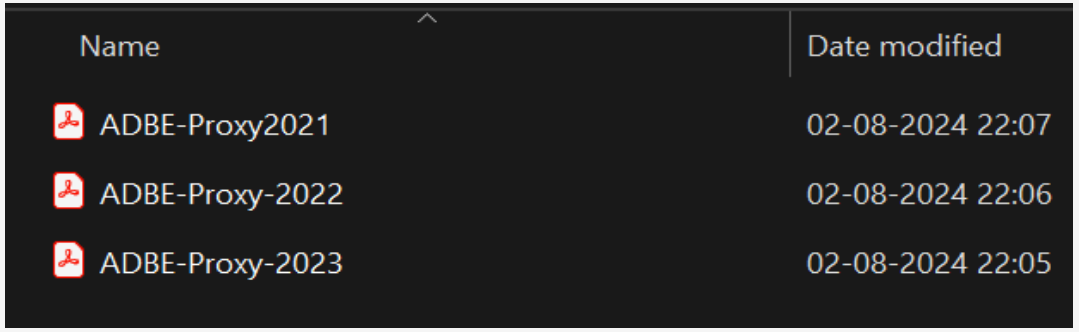
- The above subsystems can be viewed as independent modules that obey well defined interfaces
- This means that one can replace a module with another technique, so long as interfaces are respected

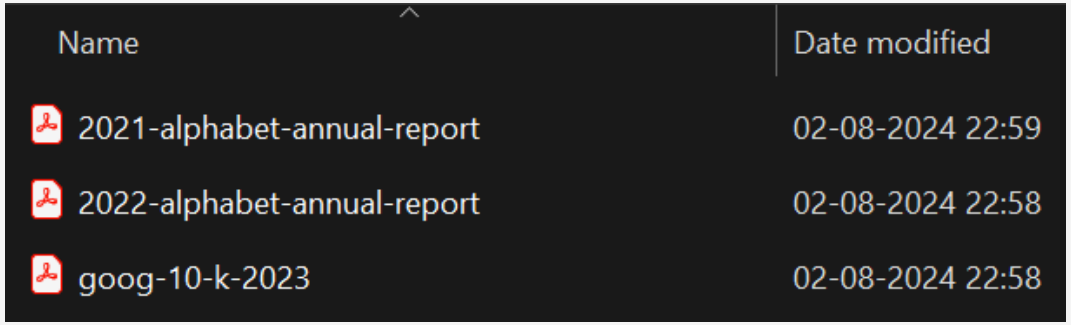
RAG Case Study #1 – Financial Analysis

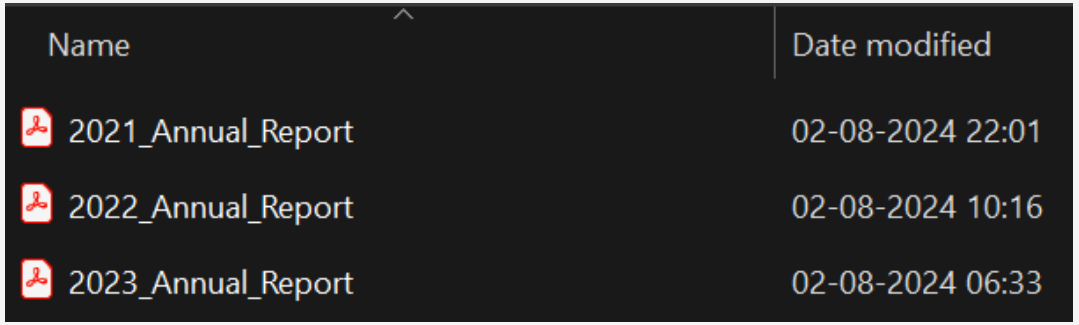
- We are given a dataset that has annual reports of Adobe, Google and Microsoft.
- We are required to build a Copilot Assistant, that takes a question from the user and provides the best possible answer.
  - In our first attempt, we will build this using a naïve approach
- This product is a competitive analyzer that can review different competitors and provide vital insights
- You can build an end to end product by including suitable UI and also graphics.
- With multimodal LLMs (e.g. Microsoft Phi-4 MM) it is possible to extend the problem to chat with charts and other visual components.

Dataset Organization









Code Walkthrough

- Ingesting
  - Preprocessing and Loading
  - Chunking (Strategy, Hyper parameters choices)
  - Embedding
  - Ingesting in to vector store

- Retrieving
  - Number of chunks to retrieve
  - Metadata based filtering
  - Formatting the chunks

- Generating
  - Prompting

Splitting/Chunking

- PDF Documents may contain images and tables besides text data.
- Chunking them with character or recursive character splitter is not enough.
- Splitting source code (e.g. Python, JS and so on) requires an understanding of the programming language.
- In such cases, a sophisticated semantic approach to chunking is required.

Refer: RetrievalTutorials/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb at main · FullStackRetrieval-com/RetrievalTutorials · GitHub

Different Levels of Chunking: Text Splitters

- Character based splitting
  - Character block size, overlap
- Recursive Character Splitting
  - Set of delimiters, split on delimiter and apply character splitting
- Document Specific Splitting
  - Specify delimiters specific to document type. E.g. For Python code, first split at class level, then function etc
- Semantic Chunking
  - At a high level, this splits into sentences, then groups into groups of 3 sentences, and then merges one that are similar in the embedding space.
- Agentic Chunking
  - Use an LLM to chunk by prompting it appropriately

Text Splitters References

Ref: https://python.langchain.com/docs/how_to/#text-splitters

Ref: langchain/libs/langchain/langchain/text_splitter.py at 9ef2feb6747f5a69d186bd623b569ad722829a5e · langchain-ai/langchain · GitHub

Ref: https://python.langchain.com/docs/how_to/recursive_json_splitter/

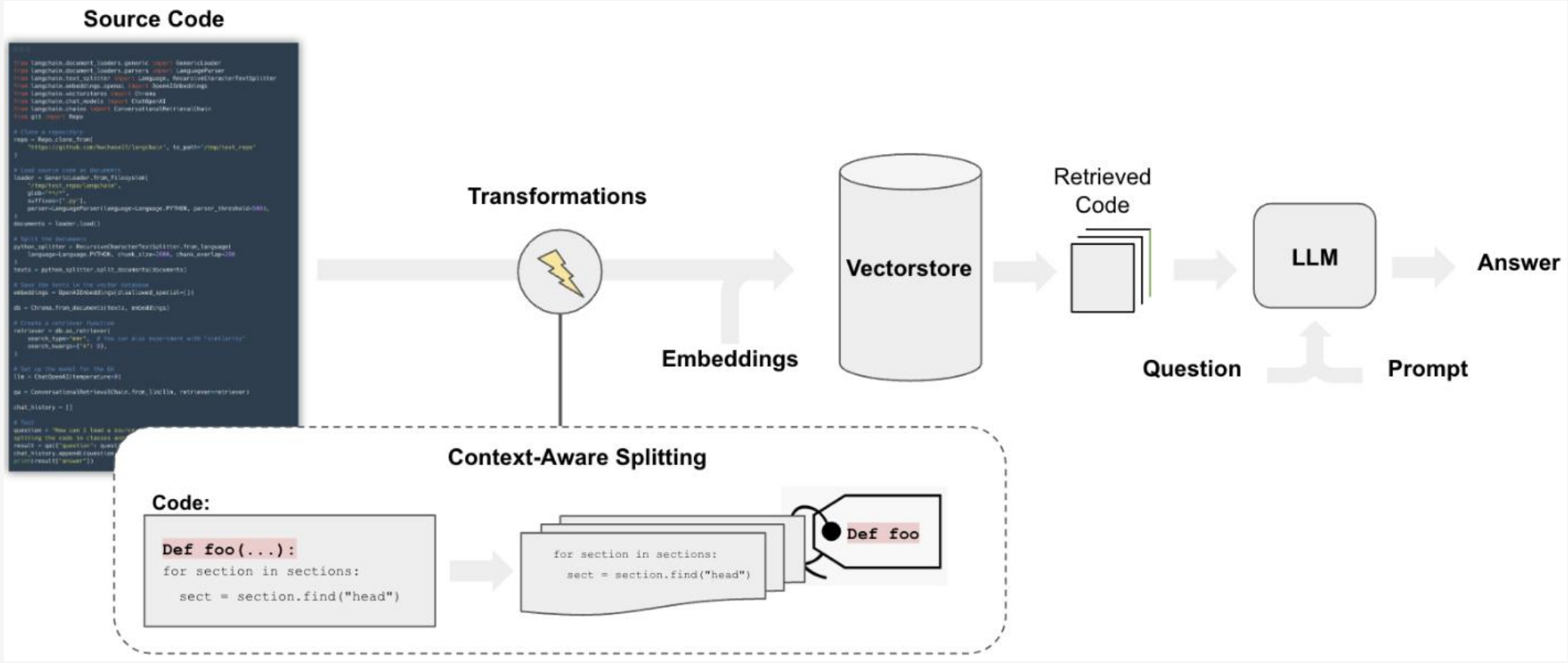
Chunk Visualization Demo

Refer: https://huggingface.co/spaces/m-ric/chunk_visualizer

RAG Case Study #2: Chat with source code

- Ask the RAG to list specific functions (searching for certain functions)
- Get the function and ask the LLM to code review it
- Get some test cases from LLM for the specified classes
- Ask the LLM to performance optimize the given module
- Refactor some code using the LLM

RAG over code



Query LangChain docs: RAG on Markdown

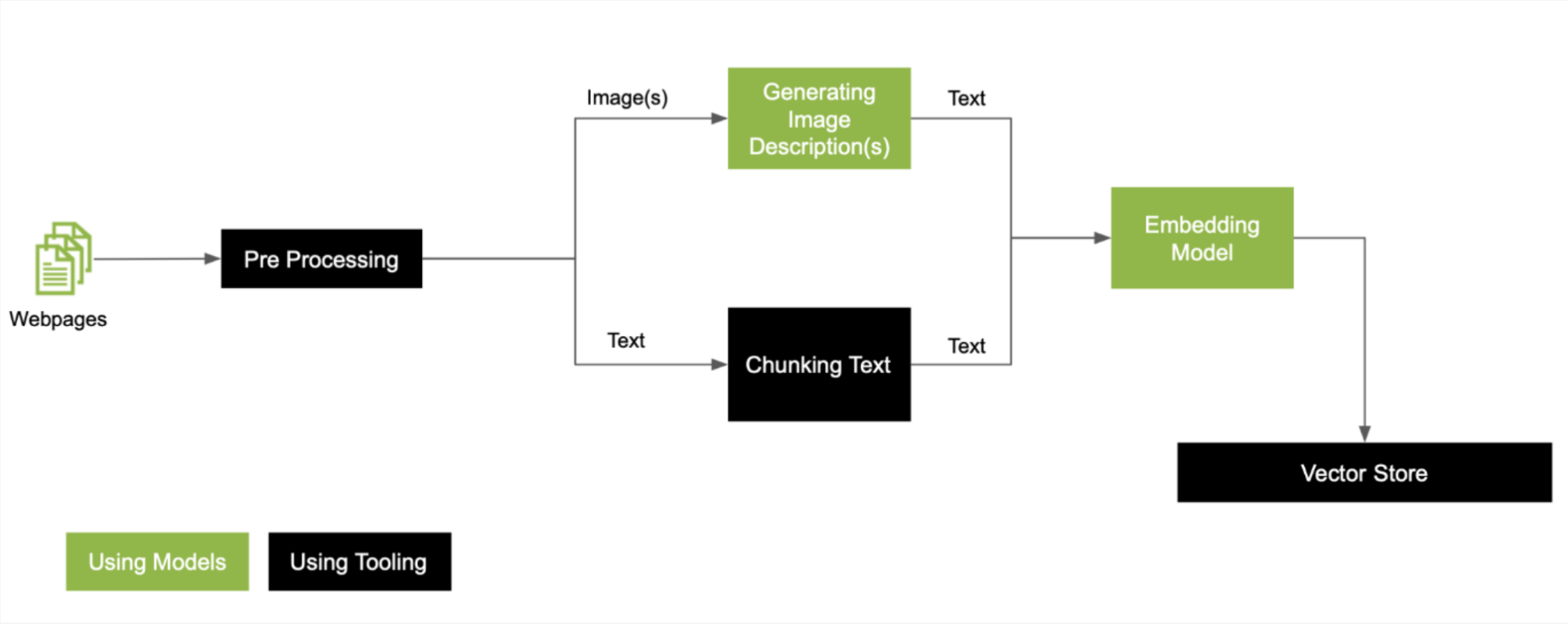
See the demo and discussions

- Generate simple agents
- Create a RAG for Multiquery retriever
- Create example of Web base loader
- Benchmark it with http://chat.langchain.com

RAG and Agents

See demo and discussions

Multimodal RAG: Preprocessing



Advanced RAG: Refer slide deck on Advanced RAG

Suggested Exercise

- Implement RAG with multiple advanced RAG techniques, each as an agent, you can optionally include Acrobat Assistant as an agent
- Write a judge agent that picks the best
- You ingest module is the same, retriever also same
- Use the best model for other downstream tasks

Key Behaviors of Agentic RAG

Agent behavior during retrieval is **adaptive and iterative**

🔁 **Query Rewriting**: Reformulates based on user need or retrieved results

🧩 **Source Selection**: Chooses best tools or indexes for the query type

🔍 **Chained Queries**: Decomposes complex tasks into sub-questions

🧠 **ReAct Loop**: Agent reasons → acts (retrieve) → observes → adjusts

🛠️ **Tool Use**: Combines search APIs, retrievers, summarizers, filters, etc.

Example: Agentic Web Retrieval

User: “Summarize latest battery technology trends in EVs”

Thought: I should search both news and research databases

Action: Run query “2024 EV battery innovations” in news tool

Observation: Results are too general — refine query

Action: Search “lithium-sulfur vs solid-state 2024 EVs”

Summarize: Aggregate into a single, coherent update

Reflect: Is the summary complete? If not, loop again

Patterns in Agentic RAG

🔍 Reflection-Guided Retrieval

- Agent critiques the initial response
- Refines or augments queries based on missing information or quality gaps

🤖 Multi-Agent Document Workflows

- Different agents handle retrieval, summarization, validation, and synthesis
- Example: Retriever → Critic → Summarizer → Integrator

📑 Retrieval Planning & Self-Evaluation

- Agent breaks down high-level query into sub-queries
- Evaluates retrieved results before proceeding to answer

🧠 Memory-Enhanced RAG

- Agents store useful facts and summaries to memory for reuse
- Enables contextual continuity across multiple queries or tasks

Comparison

Feature	🧱 Traditional RAG	🤖 Agentic RAG
Retriever Logic	Static / pre-trained	Dynamic / context-aware
Query Formulation	Single-shot, user query only	Iterative, agent-refined queries
Tool Use	One retriever, fixed method	Multi-tool orchestration (APIs, DBs, filters)
Reasoning	Minimal, LLM processes one chunk	ReAct-style loops (Thought → Action → Observe)
Adaptation to Gaps	Limited	Reflection & retry when info is missing
Memory Integration	Rare / outside of loop	Structured memory to retain insights
Use Case Fit	Simple Q&A	Complex, multi-turn, evolving tasks

Looking Ahead

- **Agentic AI** vastly expands the application space of Generative AI.
- Several Adobe products benefit from its applications – we look forward to applying the principles to our respective products
- **MCP** takes this even further – allows dynamic discoverability and effective integration of tools, databases, prompts with MCP clients and hosts
  - Applicable to all our products where we support Web Service endpoints
  - ColdFusion can help developers building MCP Servers with a few CFML tags
- Exciting times for AI developers, **Thank You!**