

LLM-RAG FROM SCRATCH: BUILD YOUR OWN OPEN- SOURCE AI CHATBOT

Mi. 10. September 2025

ETH Zürich, Gloriastrasse 35



WORKSHOPTAGE.CH



Ornella Vaccarelli

LLM-RAG from Scratch: Build Your Own Open-Source AI Chatbot

Schedule:

09:00 – Start

10:30 – Break

11:00 – Block 1

12:30 – Lunch break

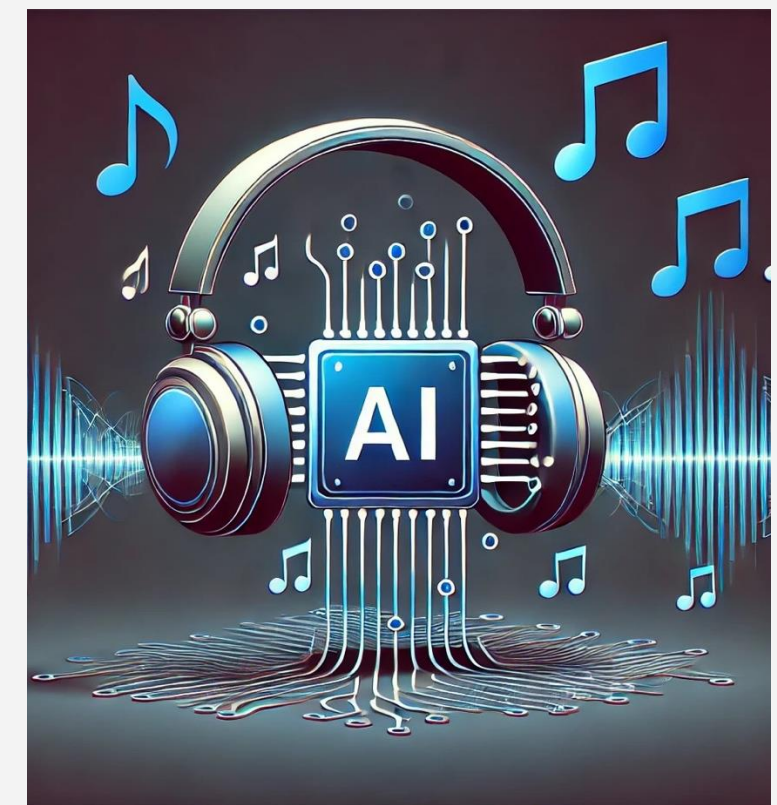
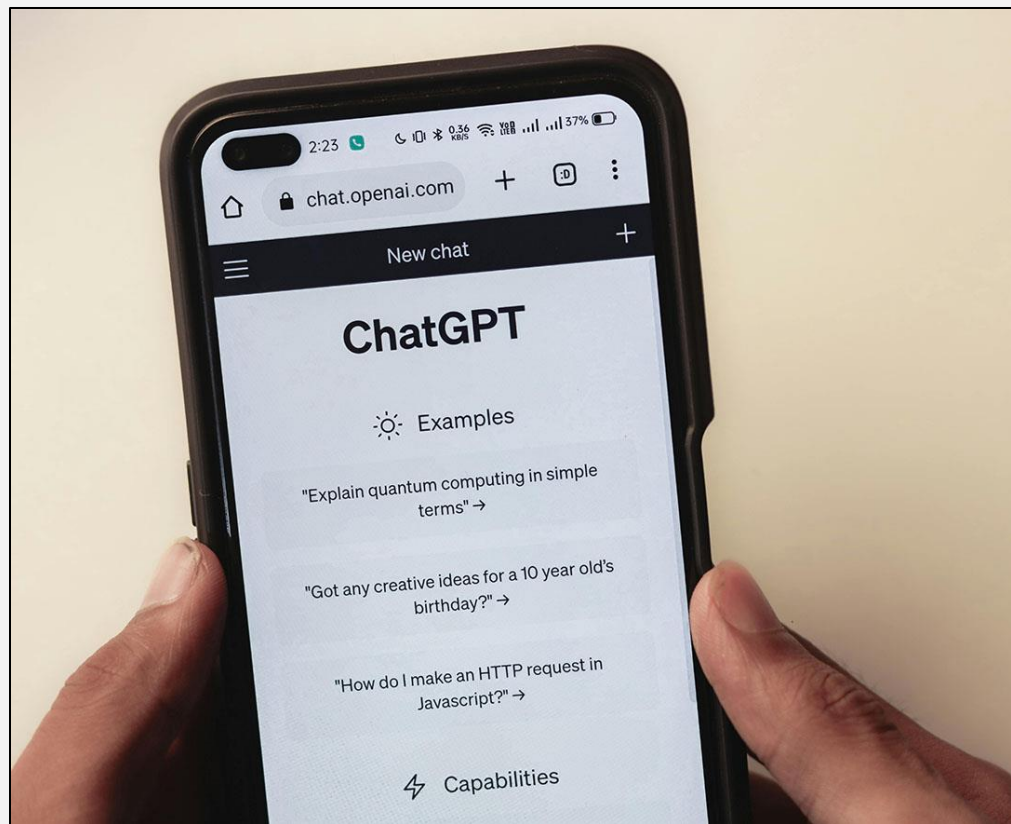
13:30 – Block 2 & 3

15:00 – Break

15:30 – Block 4 & Final discussion

17:00 – Closing

Introduction



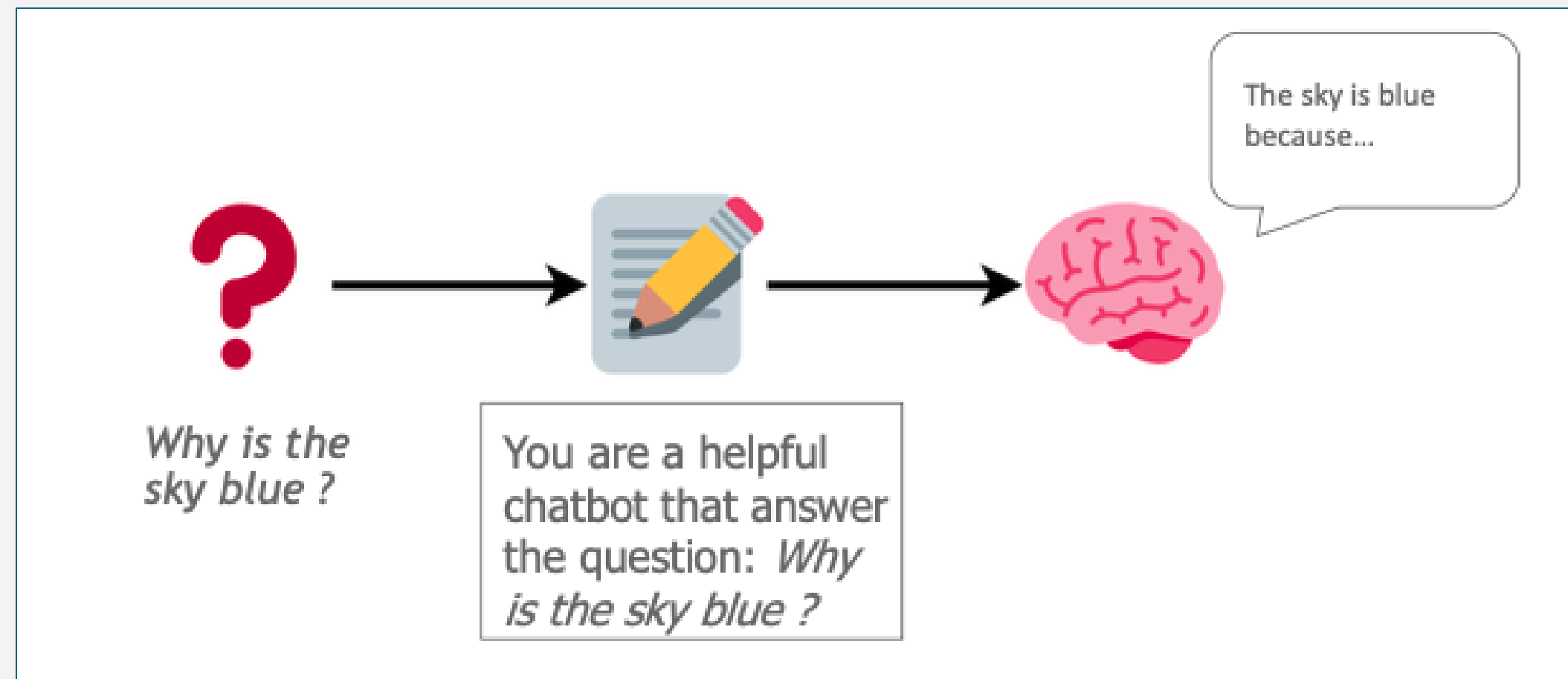
Introduction

What Is a Large Language Model (LLM)?

A **Large Language Model (LLM)** is an AI system trained on vast amounts of text that can understand context and generate human-like language.

Limit of LLMs :

- Knowledge constraints
- Tendency to “hallucinate” facts



LLM : Open-Source vs Closed-Source

Closed-Source LLMs

- Provided as a **cloud service** (API access only).
- **No access** to internal weights or training data.
- Easier to use, but less control and higher dependency on provider.
- **Examples:** OpenAI GPT, Anthropic Claude, Google Gemini.

Open-Source LLMs

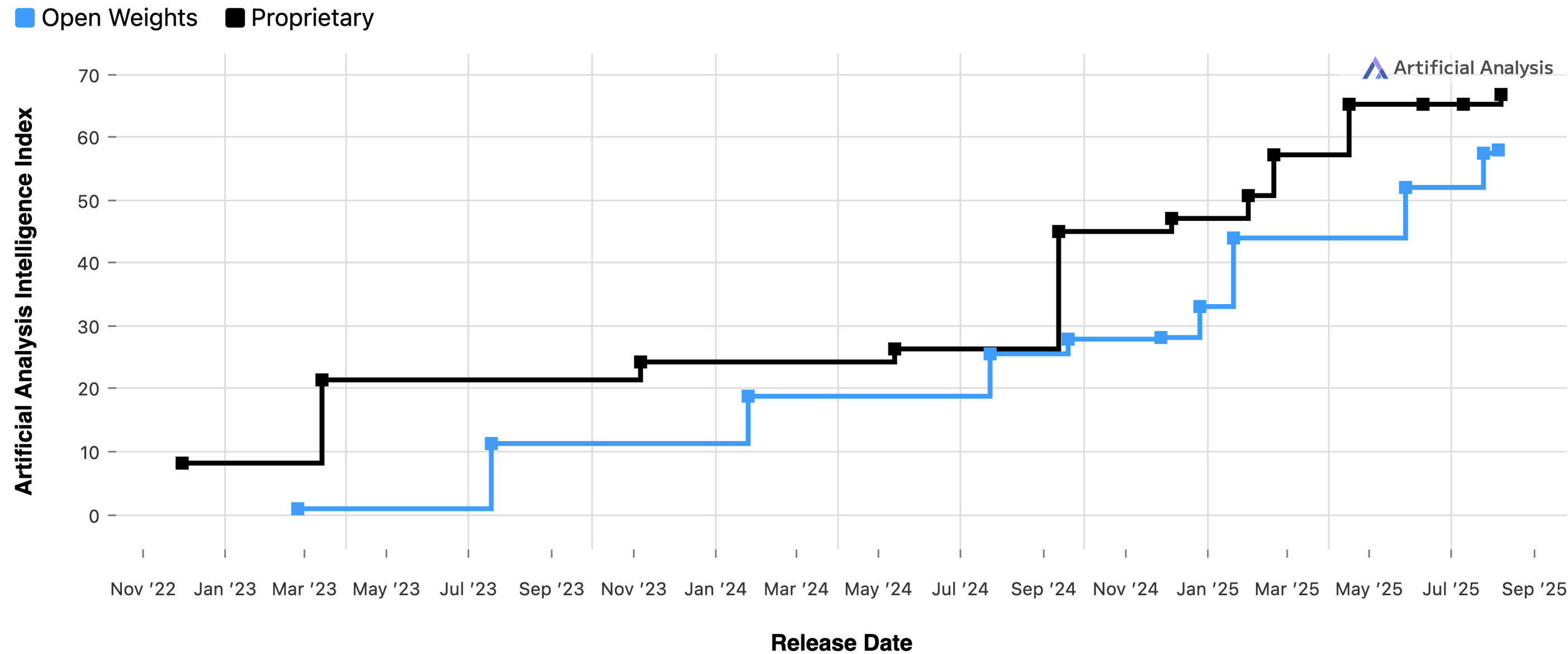
- Code and model weights are **publicly available**.
- Can run locally (full data control).
- Flexible: adapt, fine-tune, or deploy on your own infrastructure.
- **Examples:** gpt-oss, qwen3, Mistral, Gemma.

LLM : Open-Source vs Closed-Source

<https://artificialanalysis.ai/trends#progress>

Progress in Open Weights vs. Proprietary Intelligence

Artificial Analysis Intelligence Index v3.0 incorporates 10 evaluations: MMLU-Pro, GPQA Diamond, Humanity's Last Exam, LiveCodeBench, SciCode, AIME 2025, IFBench, AA-LCR, Terminal-Bench Hard, τ^2 -Bench Telecom

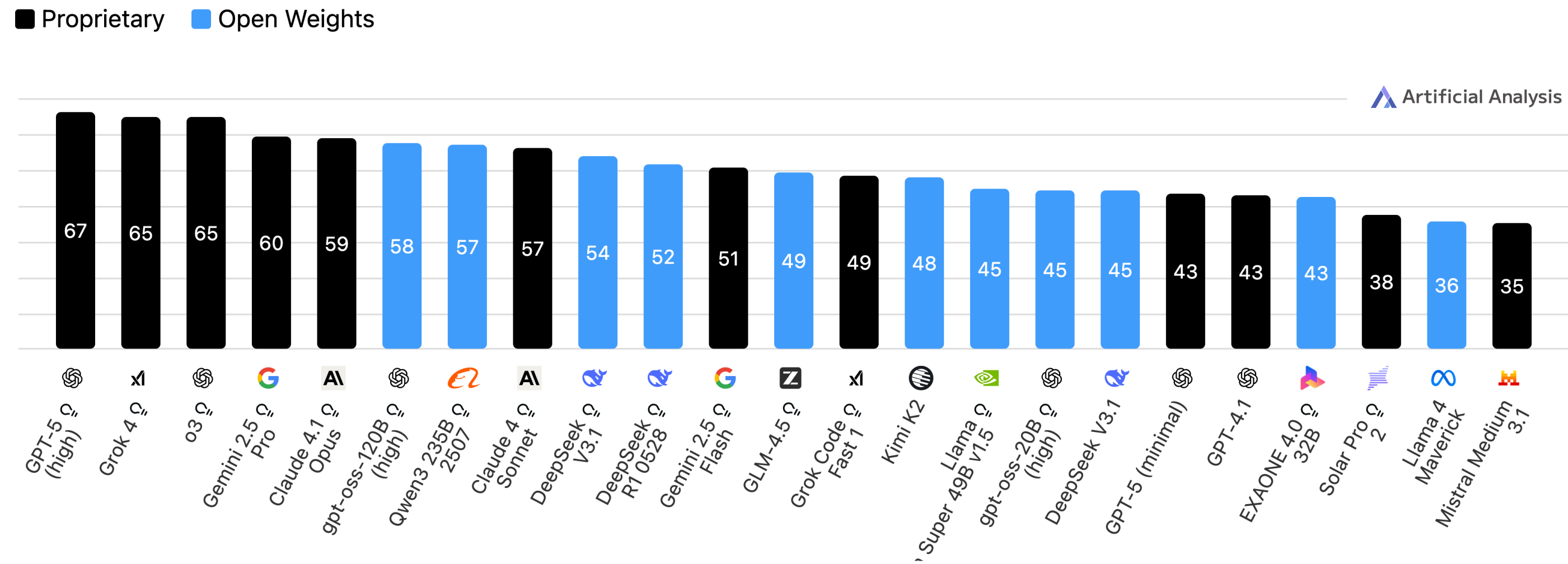


LLM : Open-Source vs Closed-Source

<https://artificialanalysis.ai/trends#progress>

Artificial Analysis Intelligence Index by Open Weights vs Proprietary

Artificial Analysis Intelligence Index v3.0 incorporates 10 evaluations: MMLU-Pro, GPQA Diamond, Humanity's Last Exam, LiveCodeBench, SciCode, AIME 2025, IFBench, AA-LCR, Terminal-Bench Hard, τ^2 -Bench Telecom



Ollama: Local Open-Source LLMs

What is Ollama?

- Open-source framework to run LLMs locally
- Runs on macOS, Linux, Windows
- Provides API via Docker (<http://localhost:11434>)

Why Ollama?

- ✓ Full control of your data (local, no cloud)
- ✓ Easy to install (`ollama pull <your-model>`)
- ✓ Supports multiple models (<https://ollama.com/search>)

1st Hands-on !!

- Go on github: <https://github.com/ovaccarelli/LLM-RAG>

LLM-RAG/

└ notebooks/

└ llm_rag_Open_Source_AI_Workshop_0_dev.ipynb

└ llm_rag_Open_Source_AI_Workshop_1.ipynb

└ llm_rag_Open_Source_AI_Workshop_2.ipynb

└ llm_rag_Open_Source_AI_Workshop_3.ipynb

└ llm_rag_Open_Source_AI_Workshop_4.ipynb

└ data/

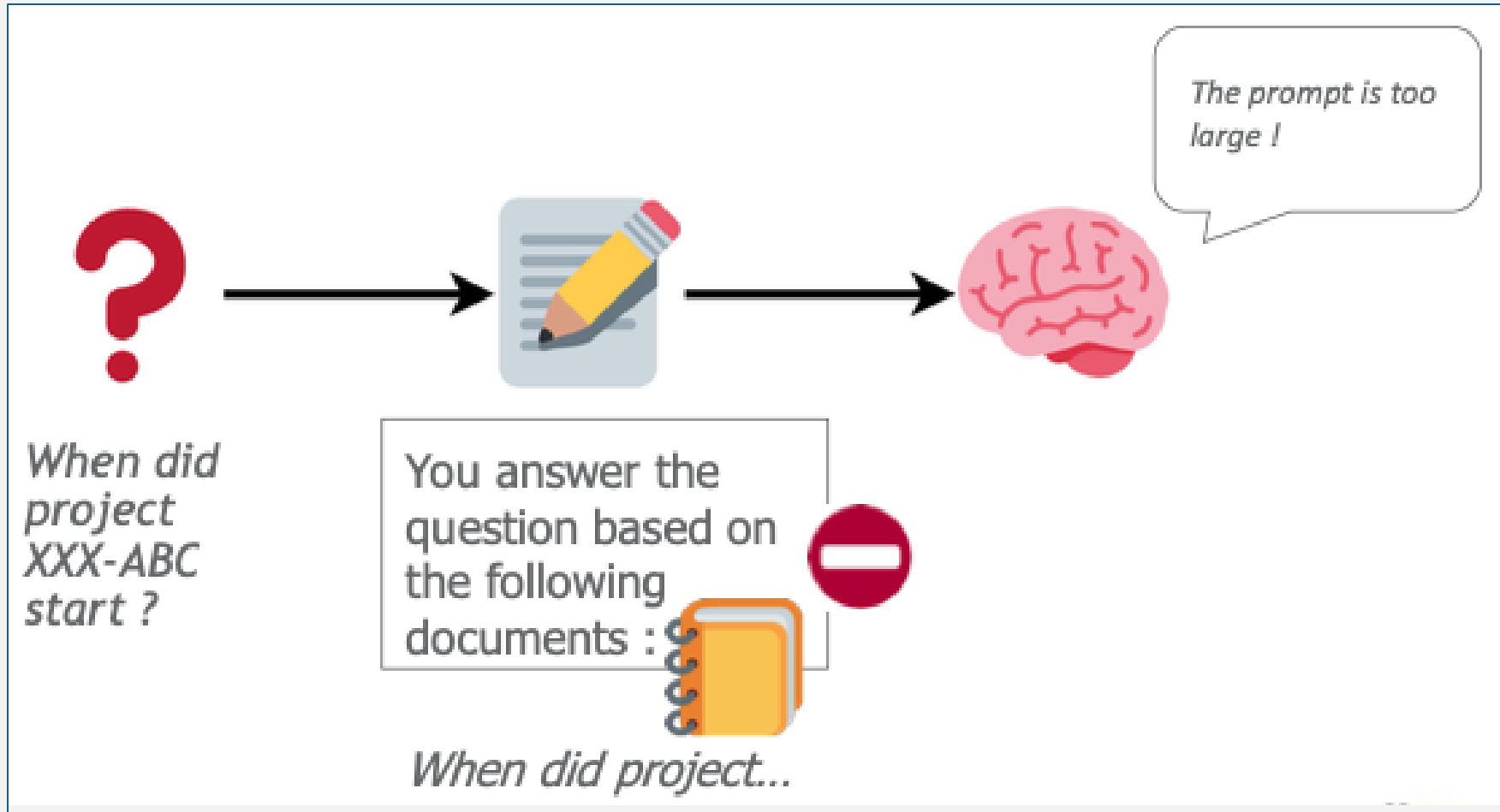
└ sample_pdf/ ← test PDF(s) for extraction

└ PDFs/ ← the main PDF(s) of our LLM-RAG

└ vectorstores/ ← saved FAISS indices

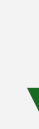
└ README.md

1. Start Your first LLM Pipeline



Context window limit (1 token ~ 4 characters)

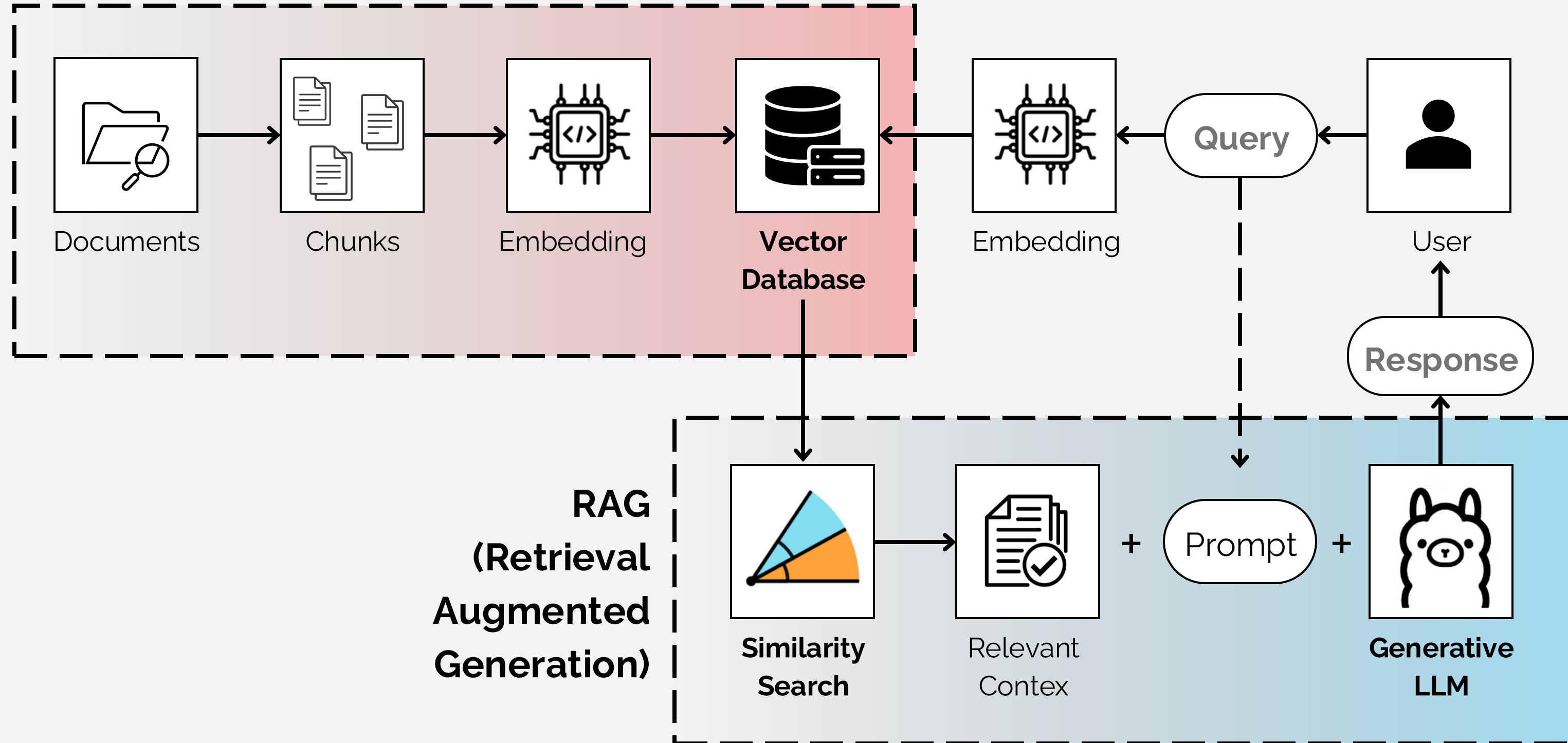
- Each LLM can only handle a **fixed number of tokens** at once.
- More tokens → higher latency + higher memory/compute use.
- Going beyond this limit → error (hallucinations) or truncated text.
- Always keep your input within the model's context size.



RAG (Retrieval-Augmented Generation)

- Only the *relevant chunks* are retrieved and injected into the prompt.
- Fewer tokens → faster, cheaper, and usually more accurate.
- Scales better with large document collections.

RAG: Architecture



2. Extract Text from a Single PDF

Retrieval-Augmented Generation for Large Language Models: A Survey

Yunfan Gao^a, Yun Xiong^b, Xinyu Gao^b, Kangxiang Jia^b, Jinliu Pan^b, Yuxi Bi^c, Yi Dai^a, Jiawei Sun^a, Meng Wang^c, and Haofen Wang^{a,c}

^aShanghai Research Institute for Intelligent Autonomous Systems, Tongji University
^bShanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
^cCollege of Design and Innovation, Tongji University

Abstract—Large Language Models (LLMs) showcase impressive capabilities but encounter challenges like hallucination, outdated knowledge, and non-transparent, untraceable reasoning processes. Retrieval-Augmented Generation (RAG) has emerged as a promising solution by incorporating knowledge from external databases. This enhances the accuracy and credibility of the generation, particularly for knowledge-intensive tasks, and allows for continuous knowledge updates and integration of domain-specific information. RAG synergistically merges LLMs’ intrinsic knowledge with the vast, dynamic repositories of external databases. This comprehensive review paper offers a detailed examination of the progression of RAG paradigms, encompassing the Naive RAG, the Advanced RAG, and the Modular RAG. It meticulously scrutinizes the tripartite foundation of RAG frameworks, which includes the retrieval, the generation and the augmentation techniques. The paper highlights the state-of-the-art technologies embedded in each of these critical components, providing a profound understanding of the advancements in RAG systems. Furthermore, this paper introduces up-to-date evaluation framework and benchmark. At the end, this article delineates the challenges currently faced and points out prospective avenues for research and development.

Index Terms—Large language model, retrieval-augmented generation, natural language processing, information retrieval

I. INTRODUCTION

LARGE language models (LLMs) have achieved remarkable success, though they still face significant limitations, especially in domain-specific or knowledge-intensive tasks [1], notably producing “hallucinations” [2] when handling queries beyond their training data or requiring current information. To overcome challenges, Retrieval-Augmented Generation (RAG) enhances LLMs by retrieving relevant document chunks from external knowledge base through semantic similarity calculation. By referencing external knowledge, RAG effectively reduces the problem of generating factually incorrect content. Its integration into LLMs has resulted in widespread adoption, establishing RAG as a key technology in advancing chatbots and enhancing the suitability of LLMs for real-world applications.

RAG technology has rapidly developed in recent years, and the technology tree summarizing related research is shown

Corresponding Author.Email:haofen.wang@tongji.edu.cn

¹Resources are available at <https://github.com/Tongji-KGLLM/RAG-Survey>

in Figure 1. The development trajectory of RAG in the era of large models exhibits several distinct stage characteristics. Initially, RAG’s inception coincided with the rise of the Transformer architecture, focusing on enhancing language models by incorporating additional knowledge through Pre-Training Models (PTM). This early stage was characterized by foundational work aimed at refining pre-training techniques [3]–[5].The subsequent arrival of ChatGPT [6] marked a pivotal moment, with LLM demonstrating powerful in context learning (ICL) capabilities. RAG research shifted towards providing better information for LLMs to answer more complex and knowledge-intensive tasks during the inference stage, leading to rapid development in RAG studies. As research progressed, the enhancement of RAG was no longer limited to the inference stage but began to incorporate more with LLM fine-tuning techniques.

The burgeoning field of RAG has experienced swift growth, yet it has not been accompanied by a systematic synthesis that could clarify its broader trajectory. This survey endeavors to fill this gap by mapping out the RAG process and charting its evolution and anticipated future paths, with a focus on the integration of RAG within LLMs. This paper considers both technical paradigms and research methods, summarizing three main research paradigms from over 100 RAG studies, and analyzing key technologies in the core stages of “Retrieval,” “Generation,” and “Augmentation.” On the other hand, current research tends to focus more on methods, lacking analysis and summarization of how to evaluate RAG. This paper comprehensively reviews the downstream tasks, datasets, benchmarks, and evaluation methods applicable to RAG. Overall, this paper sets out to meticulously compile and categorize the foundational technical concepts, historical progression, and the spectrum of RAG methodologies and applications that have emerged post-LLMs. It is designed to equip readers and professionals with a detailed and structured understanding of both large models and RAG. It aims to illuminate the evolution of retrieval augmentation techniques, assess the strengths and weaknesses of various approaches in their respective contexts, and speculate on upcoming trends and innovations.

Our contributions are as follows:

- In this survey, we present a thorough and systematic review of the state-of-the-art RAG methods, delineating its evolution through paradigms including naive RAG,



2. Extract Text from a Single PDF

Retrieval-Augmented Generation for Large Language Models: A Survey

Yunfan Gao^a, Yun Xiong^b, Xinyu Gao^b, Kangxiang Jia^b, Jinliu Pan^b, Yuxi Bi^c, Yi Dai^a, Jiawei Sun^a, Meng Wang^c, and Haofen Wang^{a,c}



^aShanghai Research Institute for Intelligent Autonomous Systems, Tongji University
^bShanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
^cCollege of Design and Innovation, Tongji University

1. PyPDFLoader

-  Simple, widely used
- Returns **Document objects** (one per page)
- Great for quick pipelines
-  Only text extraction (no figures / layout)



```
# Load the PDF with PyPDFLoader
loader = PyPDFLoader(str(pdf_path))
docs = loader.load()
```

2. PyMuPDF4LLM

-  Fast & lightweight
- Returns page dicts with Markdown text
- Good balance of speed + formatting
-  No figures/tables

```
# Load the PDF with PyMuPDF4LLM
docs = pymupdf4llm.to_markdown(str(pdf_path),
                               page_chunks=True)
```

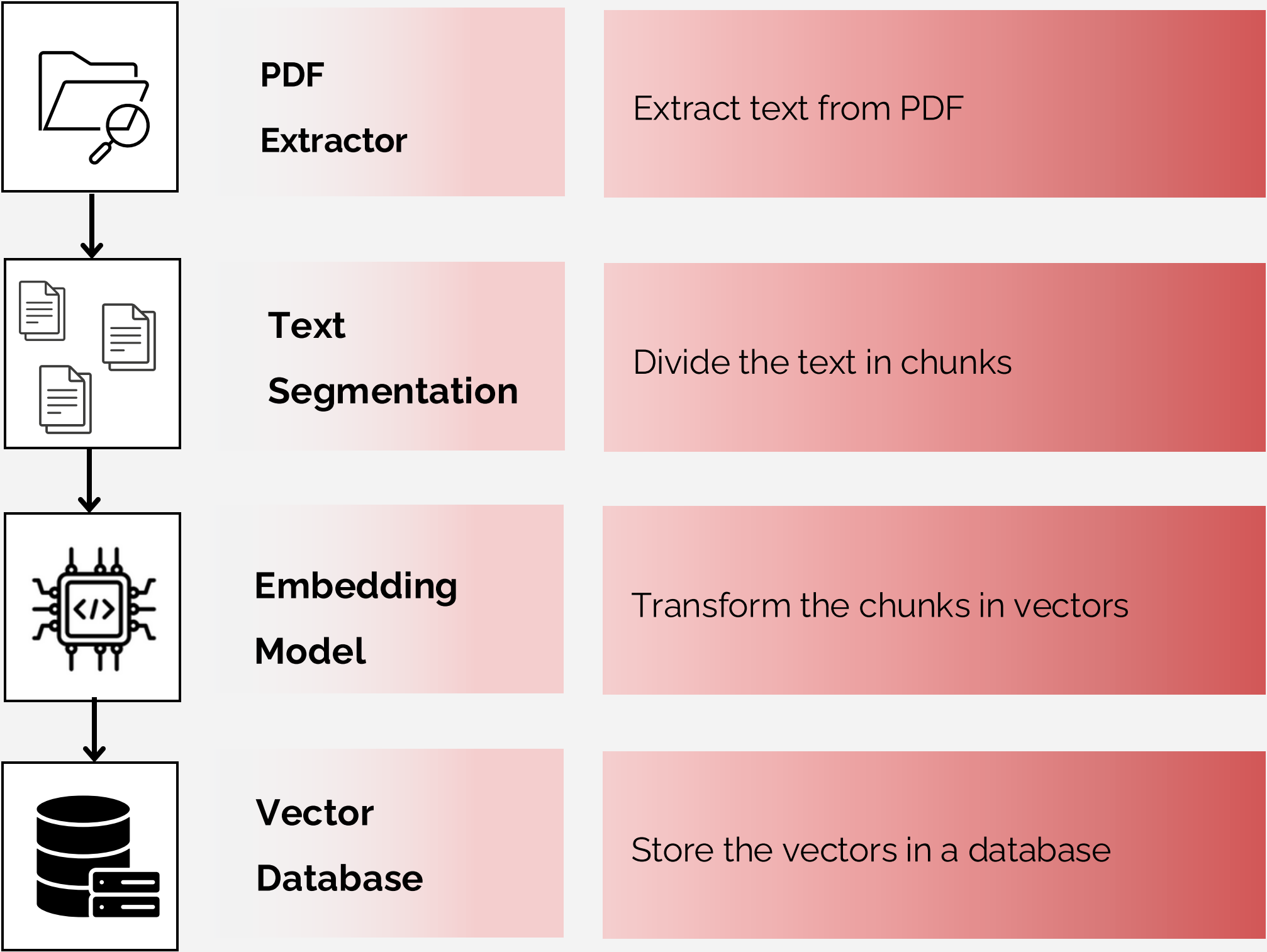
3. Docling

-  Rich parsing: text + figures, tables, layout
- Exports to Markdown (with image refs)
- Can enable OCR for scanned PDFs
-  Heavier dependency, needs OCR

```
# Load the PDF with Docling
loader = DoclingLoader(str(pdf_path),
                       export_type=ExportType.MARKDOWN)
docs = loader.load()
```

backend

3. Construct the vectorstore



LLM-RAG from Scratch: Build Your Own Open-Source AI Chatbot

Wednesday, 10 September 2025
ETH Zurich, Room F91

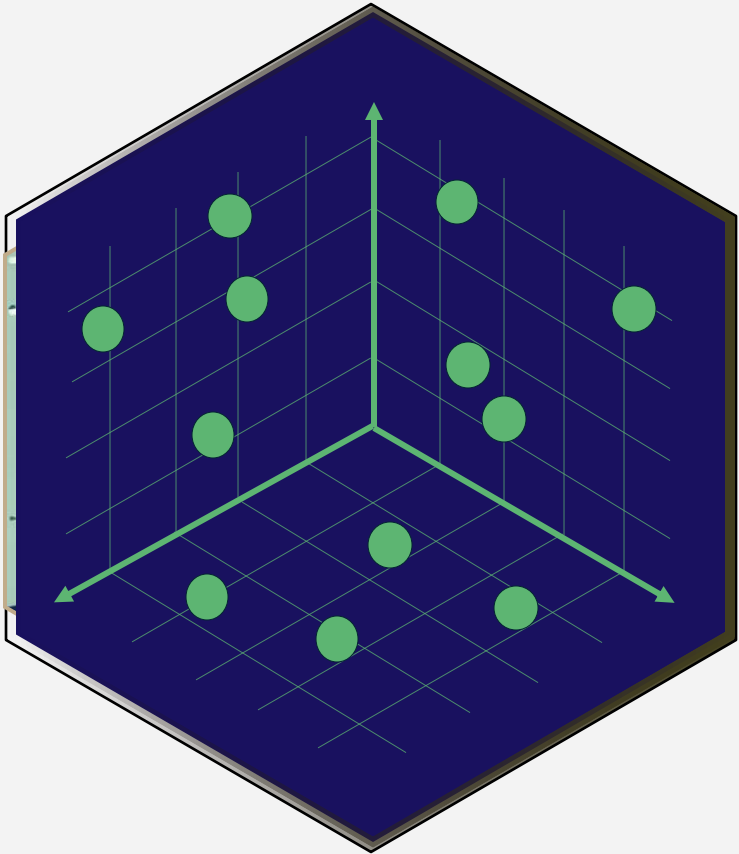
Chunk 1

This hands-on workshop offers a deep dive into the development of intelligent AI systems using the Retrieval-Augmented Generation (RAG) framework and open-source tools. Participants will learn how to build a fully functional AI chatbot that can search and understand large collections of documents to generate accurate, context-aware responses.

Throughout the session, you'll be guided through the complete RAG pipeline: from data ingestion and embedding generation to vector database integration and connection to an open-source LLM. Using frameworks such as LangChain, Hugging Face Transformers, and Ollama, you will set up each component yourself. The workshop also covers practical aspects like data handling and strategies to ensure data privacy and security during deployment.

Whether you're a developer looking to integrate LLMs into real-world applications or simply curious about how open-source AI chatbots work under the hood, this session will give you the skills and insights to build and deploy your own system.

By the end of the workshop, you'll walk away with a working open-source chatbot and a clear understanding of how to customize and expand it for your own use cases.

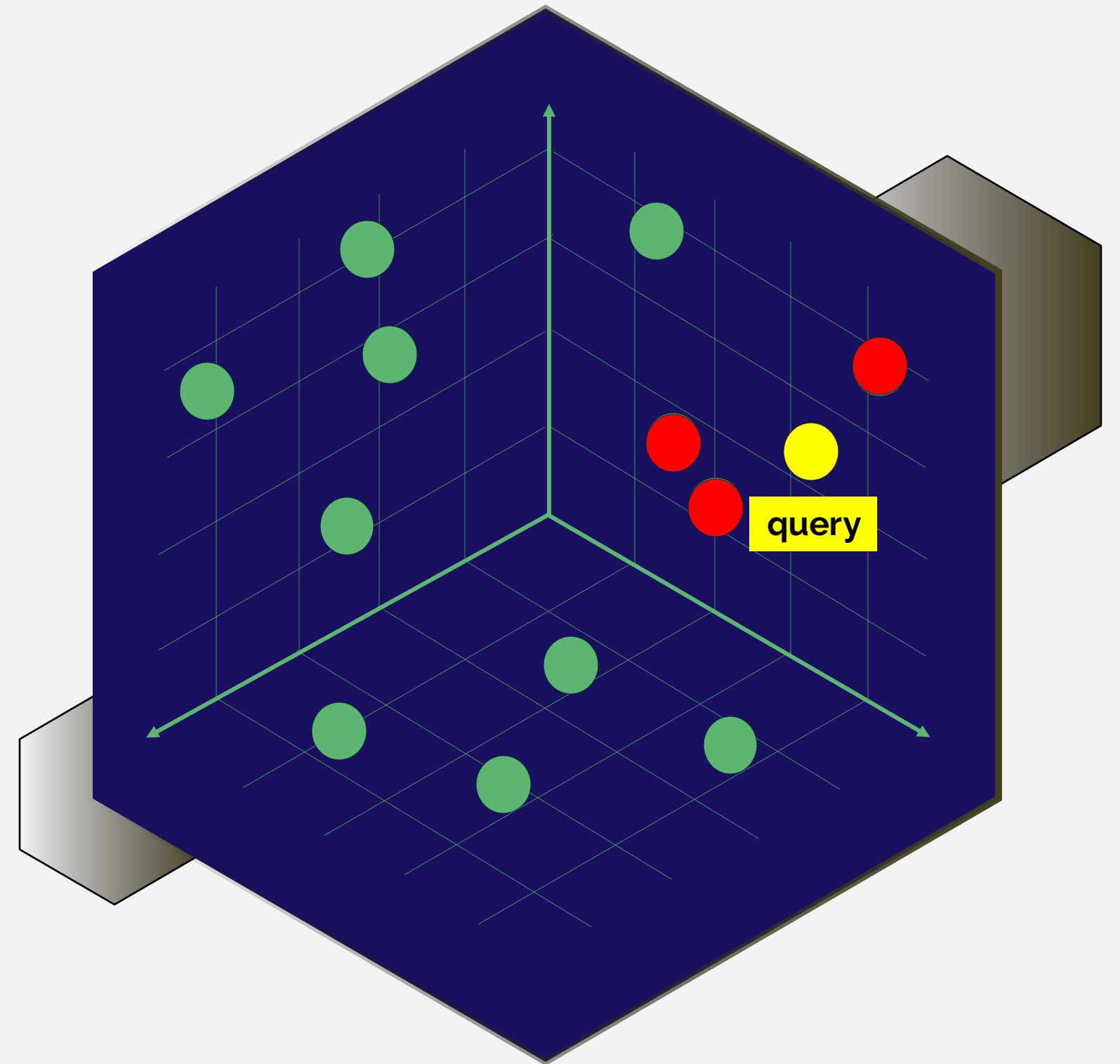


Similarity Search

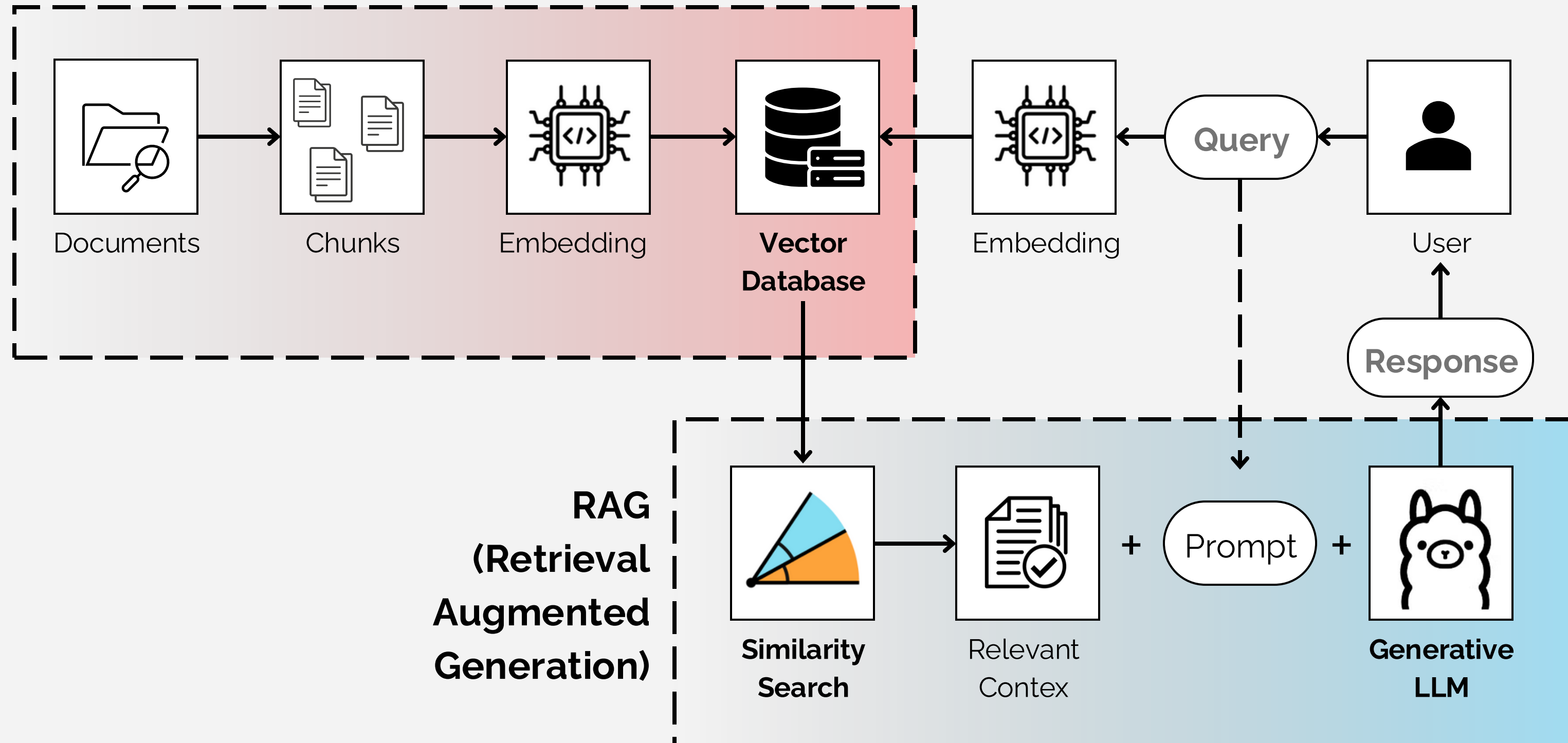
Query : also transform into a vector.

We search all the chunks "similar" to the query in order to properly answer.

- **Metric:** Cosine similarity $\text{Cos_sim}(\mathbf{q}, \mathbf{v}_i) = \frac{\mathbf{q} \cdot \mathbf{v}_i}{\|\mathbf{q}\| \|\mathbf{v}_i\|}$



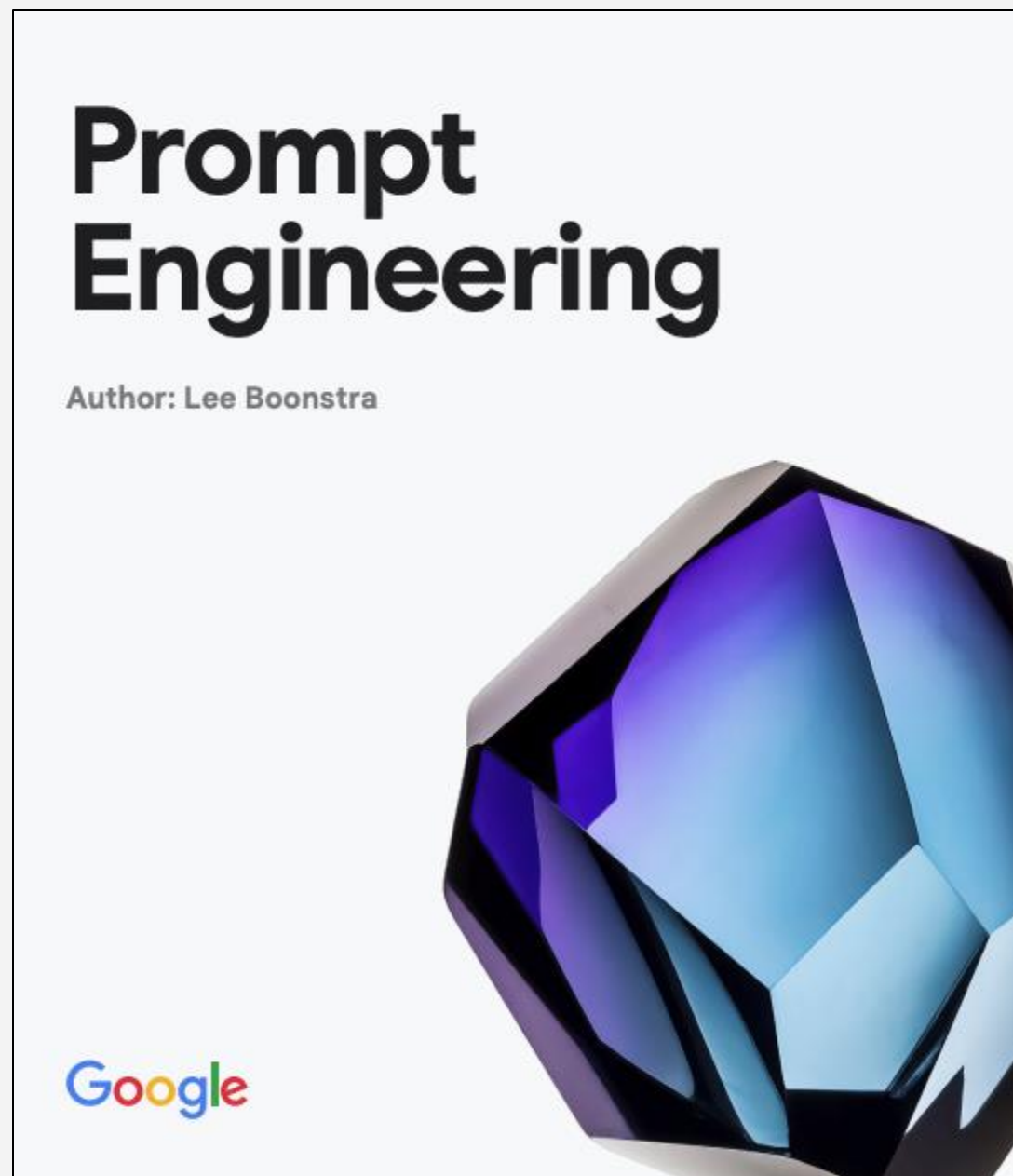
4. Final Step: Connect Retriever + LLM to Answer Questions



Prompt Engineering: Getting Better Answers

What is Prompt Engineering?

The art of formulating prompts—clear instructions, context, system rules—to guide LLM for better responses.



Prompt Engineering: Getting Better Answers

What is Prompt Engineering?

The art of formulating prompts—clear instructions, context, system rules—to guide LLM for better responses.

System VS User Prompts

- **System** : Sets the overall context and role of the LLM.
- **User** : The actual question/task: "Summarize this section in one sentence."

Best practices

1. **Set clear goals** : Use action verbs & specify format and style
2. **Be specific** : Avoid ambiguity; use clear steps
3. **Provide context** : Add background, define terms
4. **Role prompting** : "You are a financial planner..."
5. **Iterate** : Refine phrasing and structure

Prompting techniques

1. **Zero-shot** : No examples
2. **Few-shot** : Provide examples (format & content)
3. **Chain-of-Thought** : Step-by-step reasoning

Thank you !!