



# CIML Summer Institute 2024

## Retrieval Augmented Generation (RAG)

Mai H. Nguyen

# Retrieval Augmented Generation (RAG)

- **Definition:**
  - Technique to improve capabilities of LLM
- **Idea:**
  - Improve quality of text generated by LLM by incorporating additional information from an external source
- **Approach:**
  - Use a *retrieval* component to extract relevant data from an external knowledge base as context to *augment* prompt to help LLM *generate* more accurate and relevant response

# Why Use RAG?

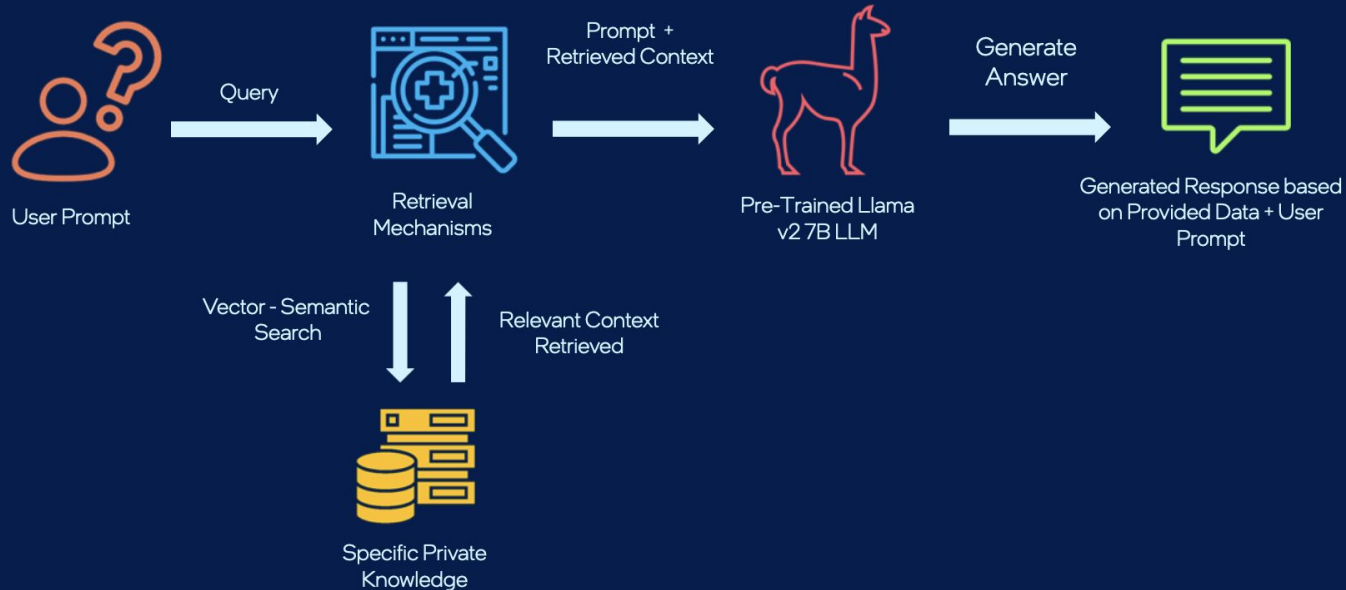
- **LLMs**
  - Responds to prompts with information from training data
- **RAG**
  - Allows LLM to access external knowledge base (e.g., company's internal database)
  - Provides most up-to-date and relevant information to LLM in generating response
  - Provides way to validate LLM's response (\*\*\*)

# How RAG Works

- User inputs prompt
- Prompt sent to retrieval system
- Retrieval system searches knowledge base and returns top relevant document chunks
- Retrieved chunks are added as context to original prompt
- Augmented prompt sent to LLM

# RAG Overview

## Retrieval Augmented Generation



Intel

# Embeddings

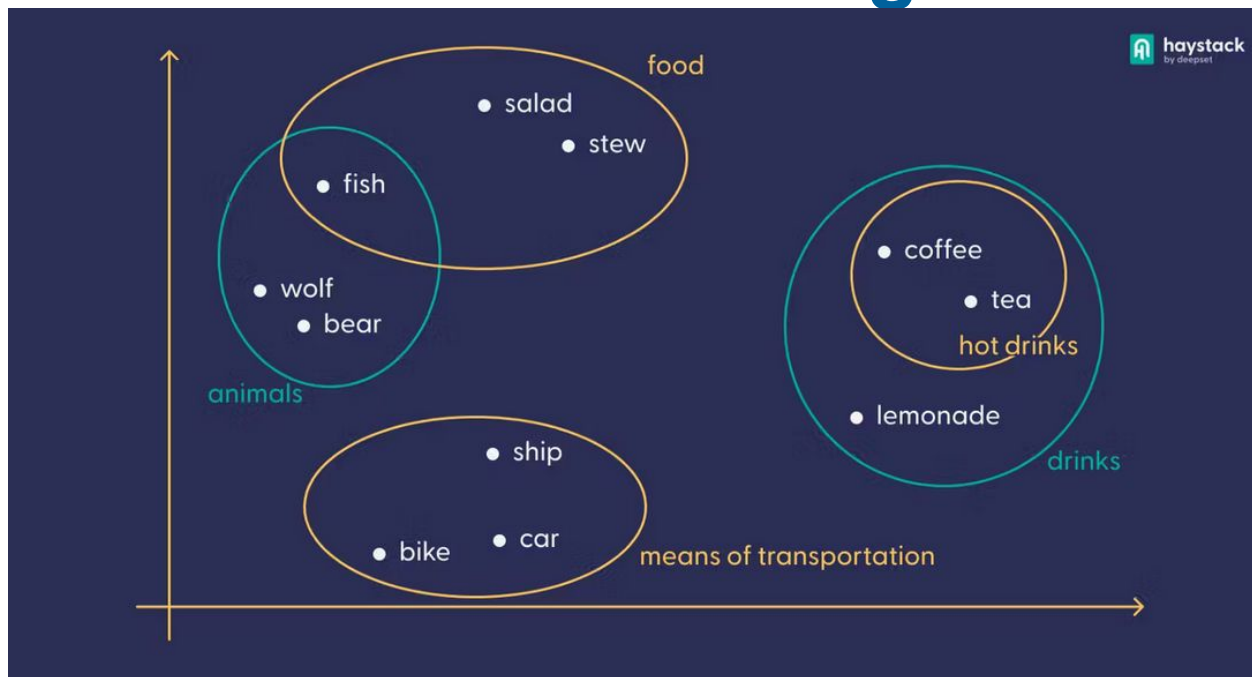
- **Text Embedding:**

- Numeric representation of text (vector of floating point numbers)
- Capture the semantics of the text
- Similarity between two embeddings indicates their semantic relatedness (cosine similarity, dot product, etc)



<https://cohere.com/blog/text-embeddings>

# Text Embeddings



<https://www.deepset.ai/blog/the-beginners-guide-to-text-embeddings>

Text embeddings allow for search and comparison between user queries and documents in knowledge base

# RAG Components

- **Embedding Model**
- **Vector Database**
- **LLM**



# RAG Components

- **Embedding Model**

- Document encoding
  - Generates embeddings for documents or text passages in knowledge base
- Query encoding
  - Generates embeddings for input query
- Some embedding models
  - HuggingFace: Universal Sentence Encoder, all-MiniLM-L6-v2
  - OpenAI: text-embedding-3-small, text-embedding-3-large (paid)

# RAG Components

- **Vector Database**

- Storage
  - Stores precomputed embeddings of documents
- Similarity search
  - Performs similarity search between query embedding and stored document embeddings to return top-k relevant documents
    - Similarity metrics: cosine similarity, dot product, etc.
  - Optimized for fast and efficient similarity search
- Some vector databases
  - ChromaDB
  - Pinecone
  - Milvus

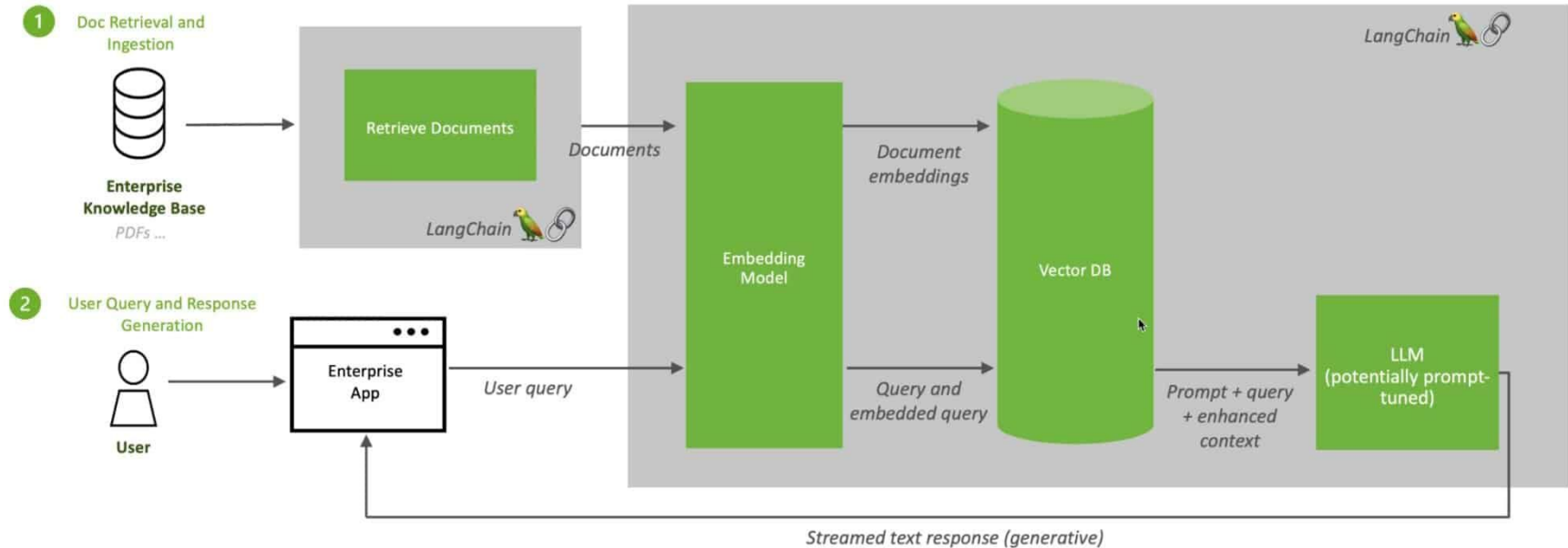
# RAG Components

- **LLM**
  - Content generation
    - Takes query augmented with retrieved content
    - Generates response to augmented prompt
  - Some LLMs
    - LLaMa3
    - Mistral
    - GPT-3.5

# RAG Overview

## Retrieval Augmented Generation (RAG) Sequence Diagram

<https://developer.nvidia.com/blog/rag-101-demystifying-retrieval-augmented-generation-pipelines/>



# RAG Hands-On

- **RAG Components**

- Embedding model: HuggingFace all-MiniLM-L6-v2
- Vector database: ChromaDB
- LLM: Mistral-7B
- LLM server: ollama

# RAG Hands-On Outline

- **Retrieval Concepts**
  - Vectorization
  - Similarity
  - Vector database
  - Chunking
- **Basic RAG**
- **RAG with LangChain**

# RAG Hands-On - Setup

- **In terminal window**
  - `jupyter-gpu-shared-llm`
    - Alias for: `galileo launch --account ${CIML_ACCOUNT} --reservation ${CIML_RESERVATION_CPU} --partition shared --cpus 4 --memory 32 --gpus 1 --time-limit 02:00:00 --env-modules singularitypro --sif /cm/shared/examples/sdsc/ciml//2024/LLM/ollama_late.sif --nv --bind /expance,/scratch,/cm --quiet`
  - Copy and paste URL to browser window
- **To check queue**
  - `squeue -u $USER`

# RAG Setup

- **In terminal window in Jupyter Lab**
  - Type: `ollama serve`
  - This starts ollama service to serve LLM
- **In another terminal window in Jupyter Lab**
  - Type: `ollama pull mistral`
  - This pulls down the Mistral model from the ollama server. This is the LLM model we will use.



# Resources

- RAG
  - <https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag>
  - [Step-by-Step Tutorial on Integrating Retrieval-Augmented Generation \(RAG\) with Large Language Models | by Novita AI | Apr, 2024 | Medium](#)
  - [ollama/examples/langchain-python-rag-document/main.py](#) at main
- Embedding model
  - <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>
- ChromaDB
  - <https://docs.trychroma.com/>
  - <https://colab.research.google.com/drive/181Kummxd8yOyRqFu8l0aqjs2aqnOy4Fu?usp=sharing>
- Mistral-7B
  - <https://mistral.ai/news/announcing-mistral-7b/>
  - <https://ollama.com/library/mistral>

# Resources

- LangChain
  - [https://python.langchain.com/v0.1/docs/use\\_cases/question\\_answering/quickstart/](https://python.langchain.com/v0.1/docs/use_cases/question_answering/quickstart/)
  - [https://api.python.langchain.com/en/latest/chains/langchain.chains.retrieval\\_qa.base.RetrievalQA.html](https://api.python.langchain.com/en/latest/chains/langchain.chains.retrieval_qa.base.RetrievalQA.html)
  - <https://python.langchain.com/v0.2/docs/integrations/vectorstores/chroma/>
  - [https://python.langchain.com/v0.2/docs/integrations/text\\_embedding/huggingfacehub/](https://python.langchain.com/v0.2/docs/integrations/text_embedding/huggingfacehub/)
- Ollama:
  - <https://www.ollama.com/>
  - [Ollama | !\[\]\(467d80e979964f7f8c752fb22248b5b7\_img.jpg\) !\[\]\(b71552d33dbf62adf5e5199a70ee02bf\_img.jpg\) LangChain](#)

# Acknowledgements

- **Hou Wan**
  - Created RAG hands-on
- **Mahidhar Tatineni**
  - LangChain RAG borrows from Mahidhar's RAG example

# Questions?

