

Introduction to RAG (Retrieval-Augmented Generation)

Mami Hayashida and Vikram Gazula

February 20, 2025

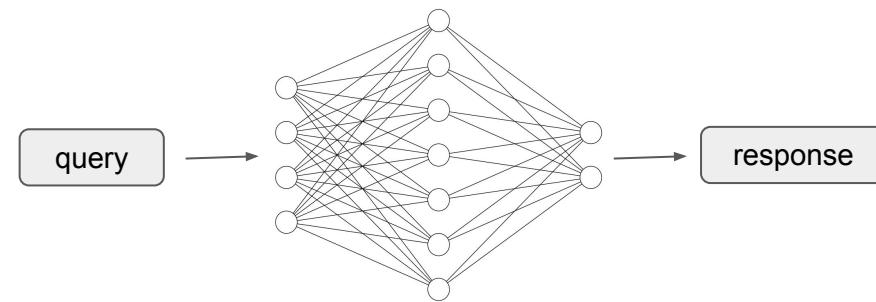
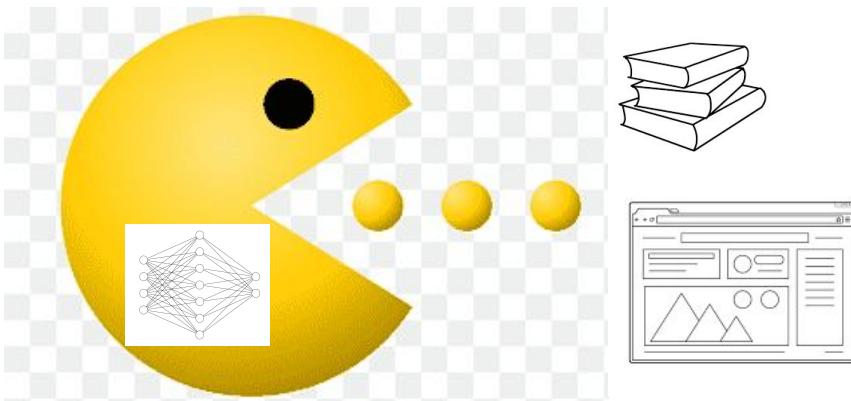
Outline

- What is LLM?
- What is RAG?
- Try running LLM + RAG pipeline locally (demo)
- Use cases
 - ACCESS-CI
 - FABRIC
- Q & A

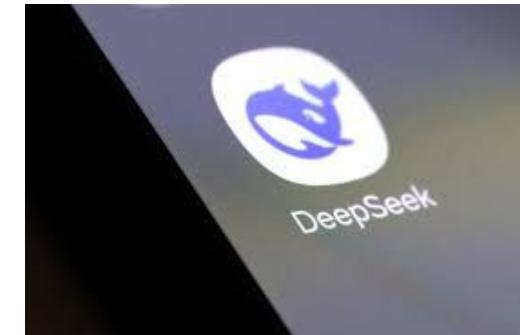
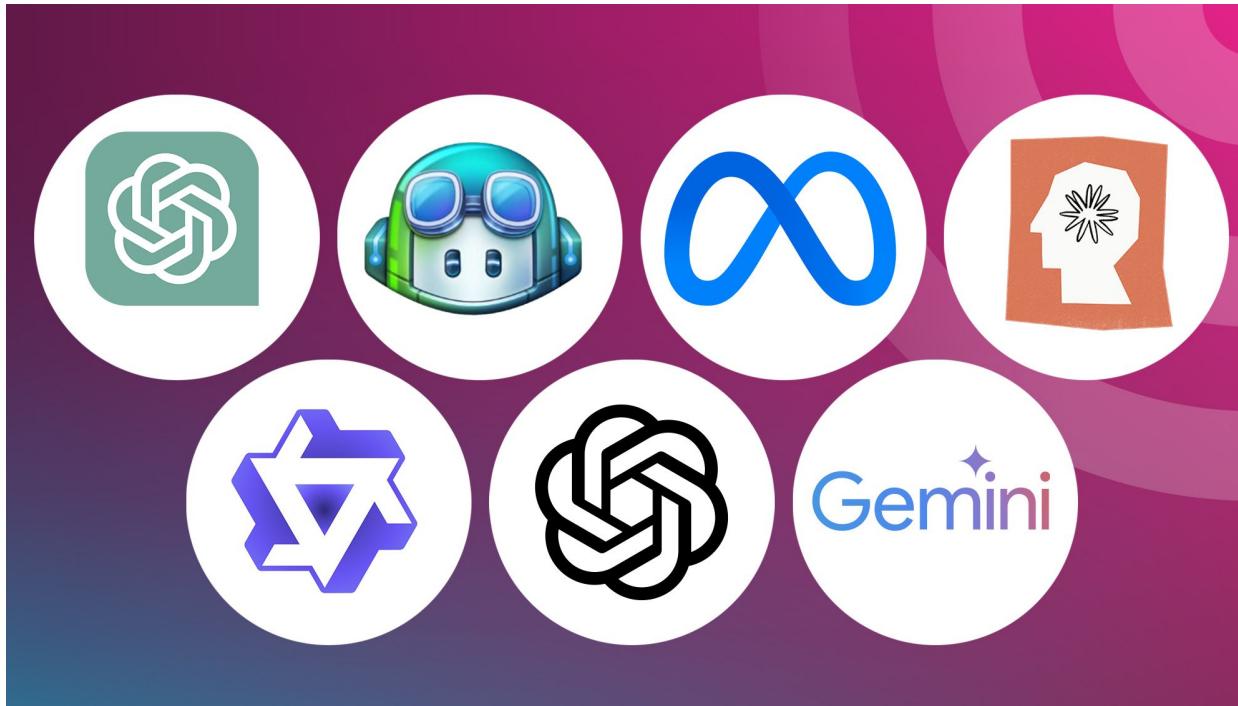
LLM

A large language model (LLM) is a type of machine learning model designed for natural language processing tasks such as language generation. LLMs are language models with many parameters, and are trained with self-supervised learning on a vast amount of text.

(https://en.wikipedia.org/wiki/Large_language_model)



Growing number of GenAI options



source:

<https://www.pbs.org/newshour/science/what-is-deepseek-heres-a-quick-guide-to-the-chinese-ai-company>

source: <https://www.techradar.com/computing/artificial-intelligence/best-langs>

GenAI shortcomings

- 1. Lack of domain-specific/custom knowledge**

GenAI shortcomings

1. Lack of domain-specific/custom knowledge

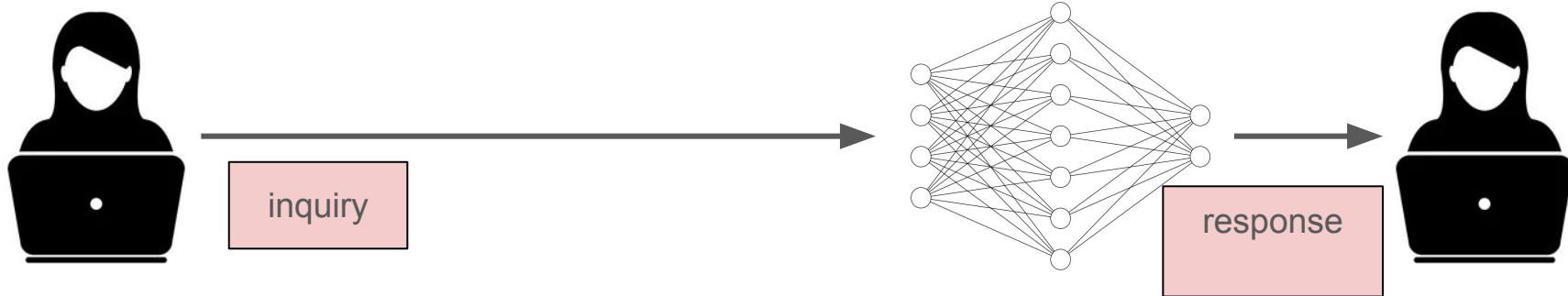


- Train a new model 😱
- Fine-tune an existing model 😕
- Use RAG 😊

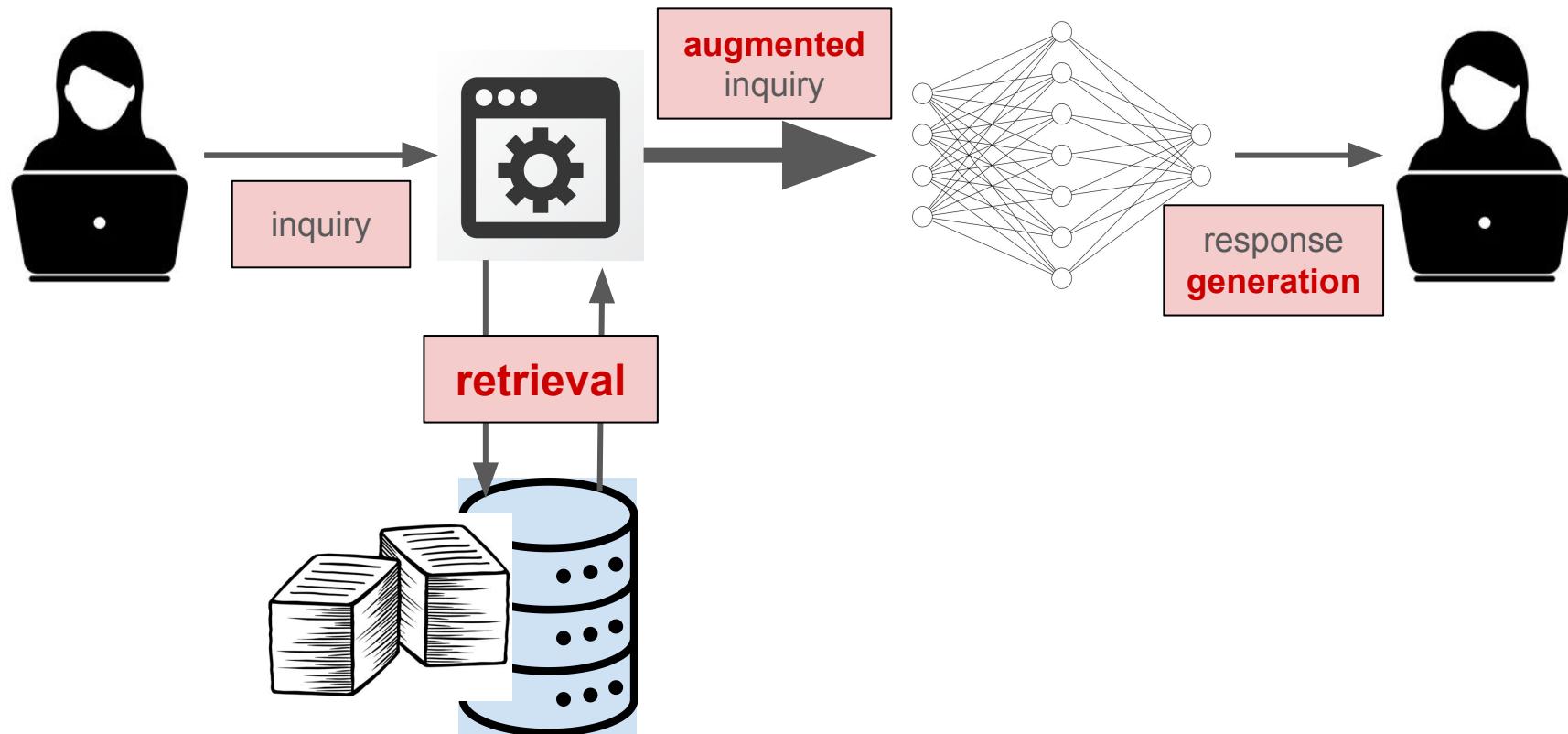
RAG

(Retrieval Augmented Generation)

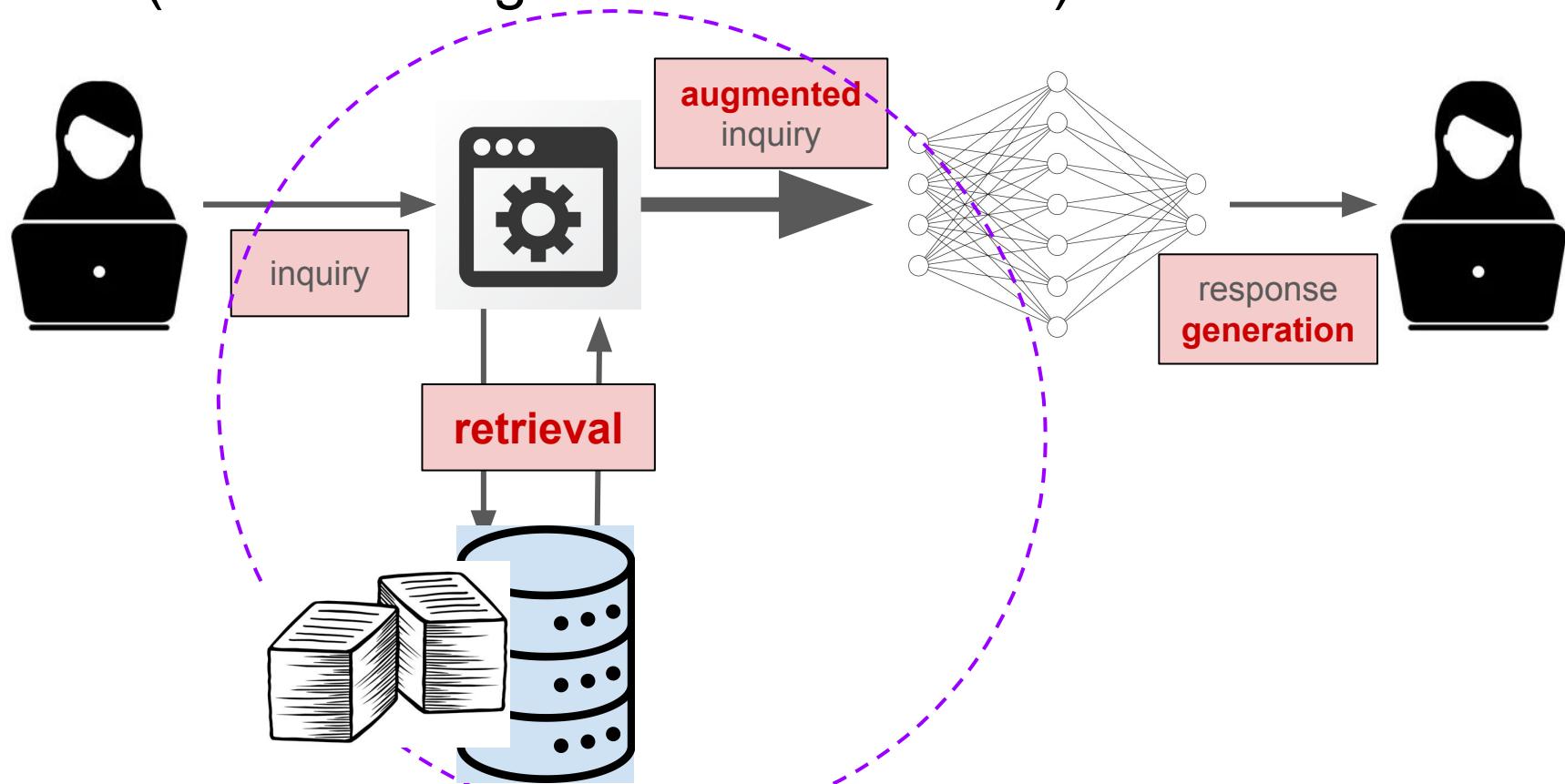
Without RAG



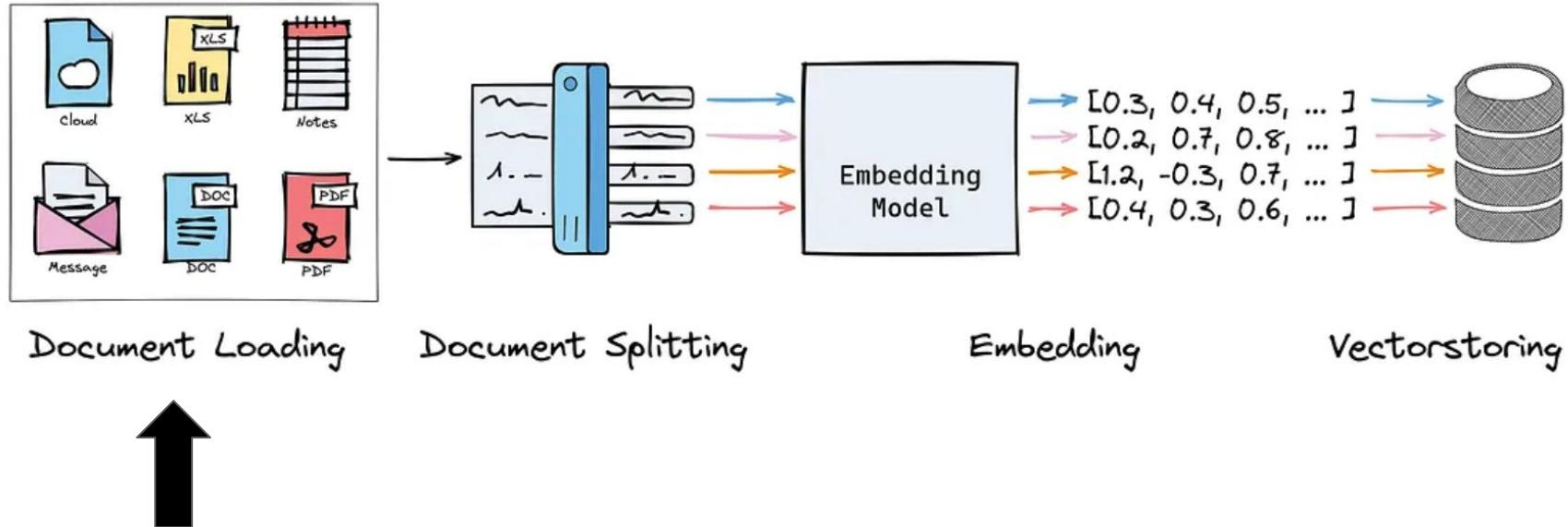
RAG (Retrieval Augmented Generation)



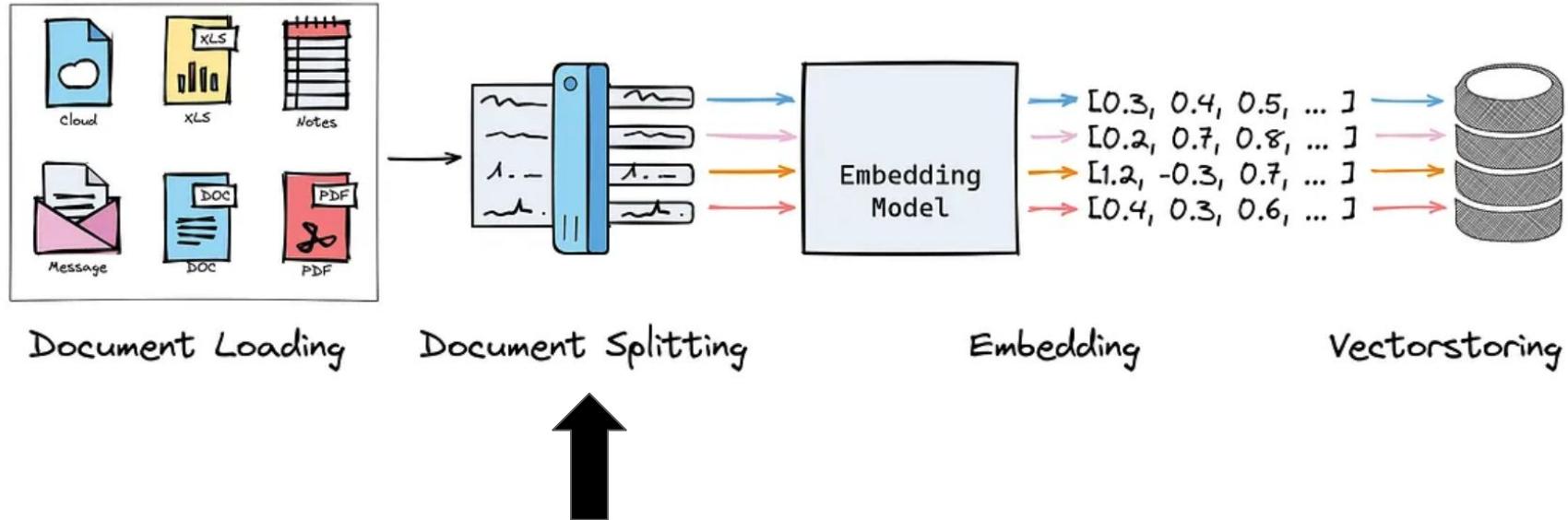
RAG (Retrieval Augmented Generation)



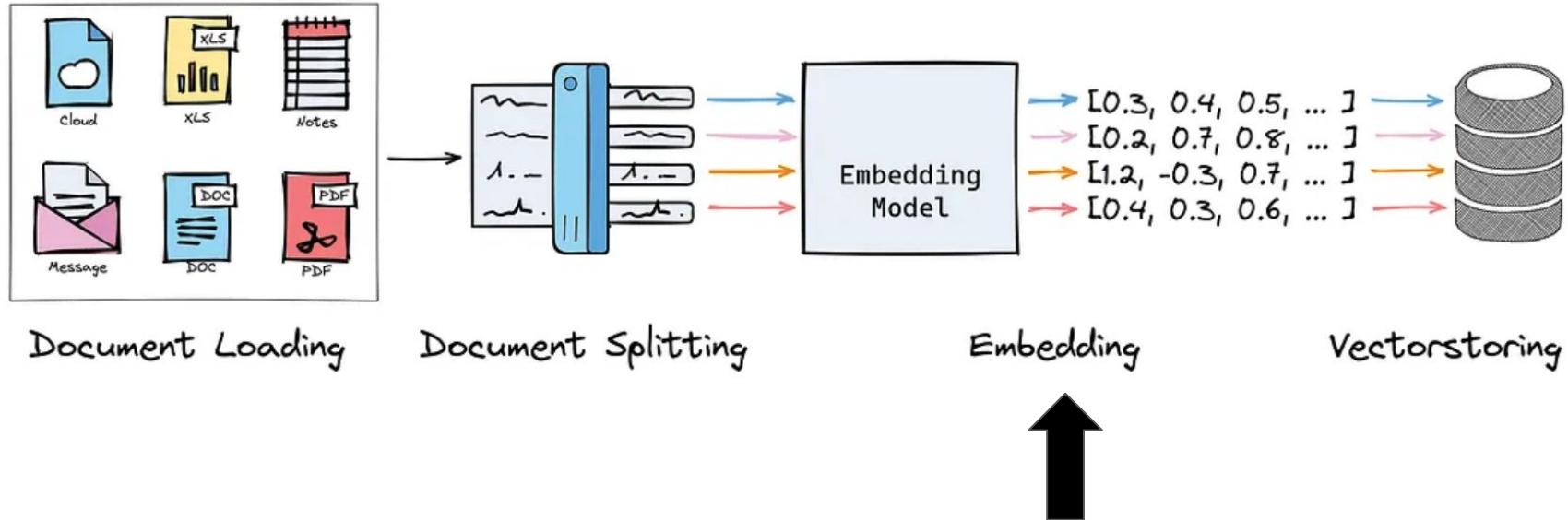
Storing your data on Vector Database: 1. Document Loading



Storing your data on Vector Database: 2. Document Splitting



Storing your data on Vector Database: 3. Embedding



Storing your data on Vector Database: 3. Embedding

The Future of Deep Sea Exploration ...
The Art of Minimalism
Urban Greening: Transforming Cities ...
Quantum Computing and Its Impact on ...
The Rise of Global Cuisine in Local ...
The Evolution of Digital Art and NFTs
Renewable Energy Innovations: Solar ...
Strategies for Enhancing Work Efficiency

Knowledge Base



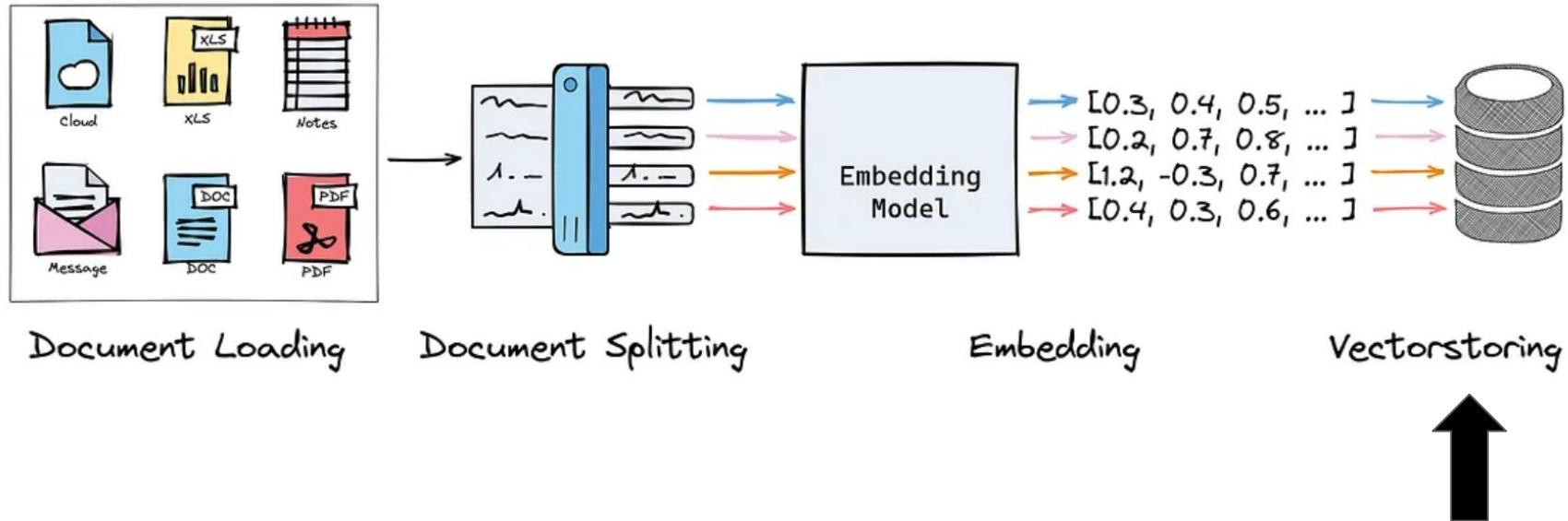
Vectors

Example: embed query



- Embedding translates each of the split text “chunks” into a high-dimensional numerical representation.
- Goal is to preserve the semantic and contextual similarities and differences.

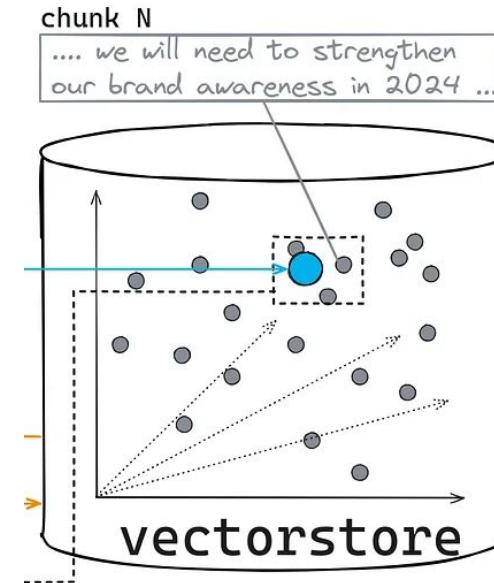
Storing your data on Vector Database: 4. Vectorstoring



Storing your data on Vector Database: 4. Vectorstoring

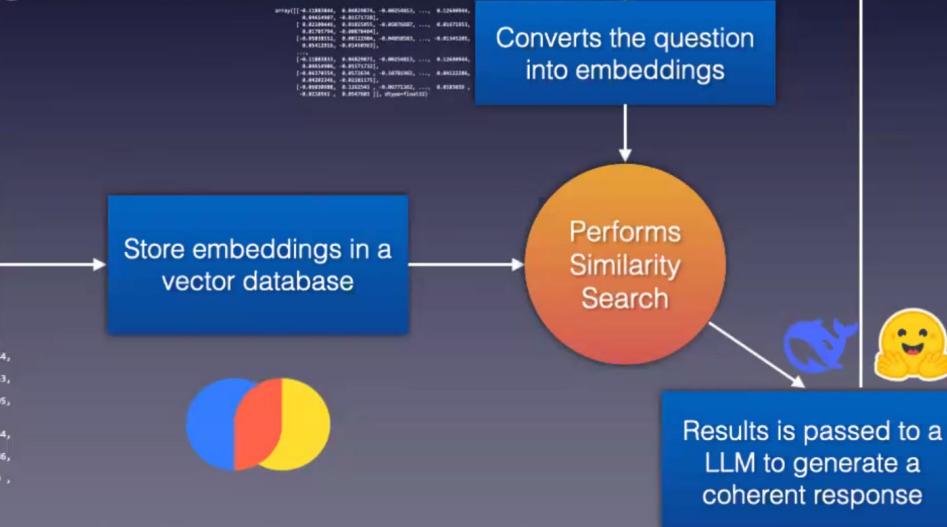
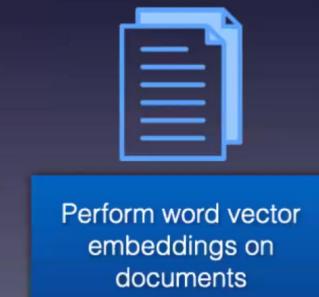
In short, Vectorstore store vectors where chunks are organized by their similarities vs. differences.

(Embedding and vectorstoring are still relatively slow.)

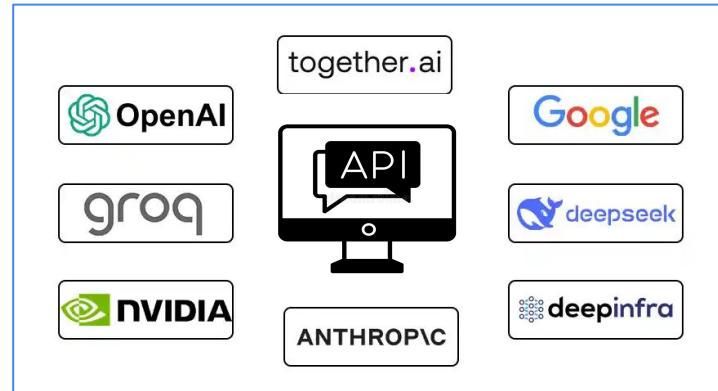
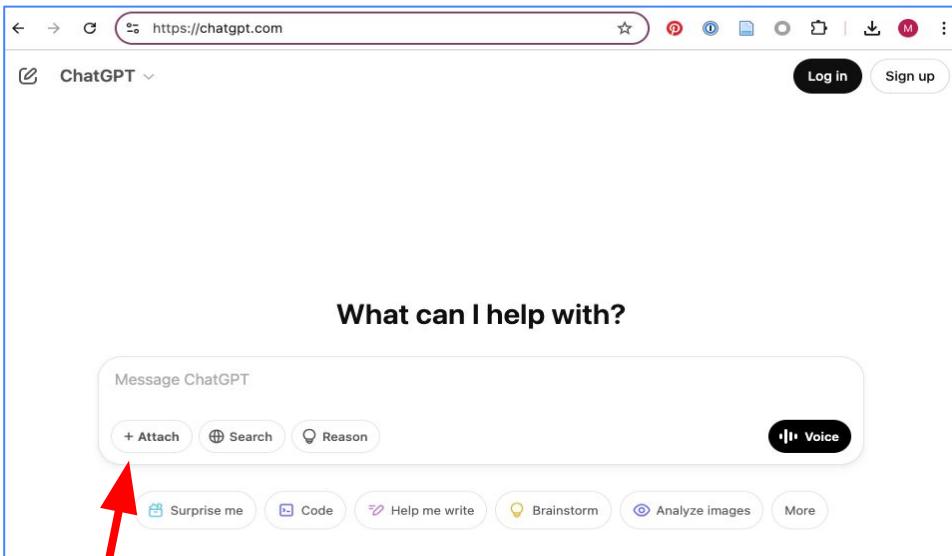


How RAG Works

Private Documents



Uploading your data to 😱



source: <https://www.analyticsvidhya.com/blog/2024/10/free-and-paid-apis/>

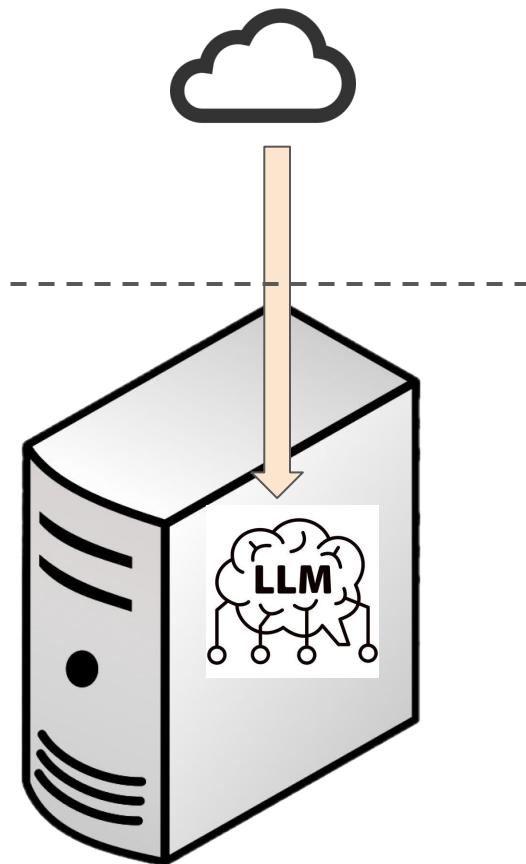


GenAI shortcomings

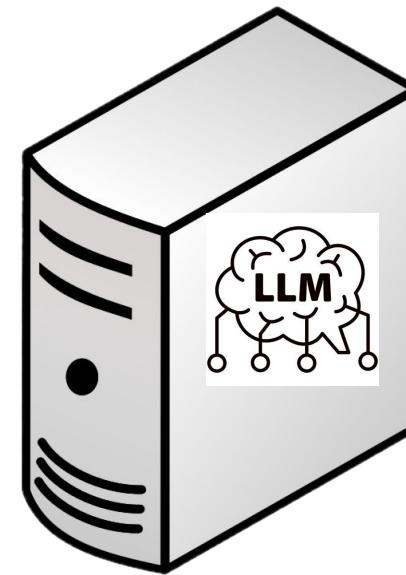
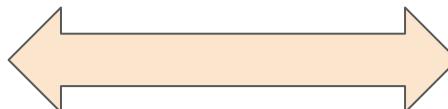
- 1. Lack of domain/specific knowledge**
- 2. Privacy/confidentiality concerns**

Running LLM + RAG Locally

Local LLM ecosystem



Local LLM ecosystem



Running your LLM Locally

- Ollama 
- LM Studio  LM Studio
- llama.cpp 

All three support a REST API, allowing you to integrate LLMs into your apps.

credit: Wei-Meng Lee

Ollama

<https://ollama.com/>



Get up and running with large language models.

Run [Llama 3](#), [Phi 3](#), [Mistral](#), [Gemma](#), and other models. Customize and create your own.

Download ↓

Available for macOS, Linux, and Windows (preview)

Ollama allows you to run open-source large language models, such as Llama 3, deepseek-r1 locally.

Amazingly simple!

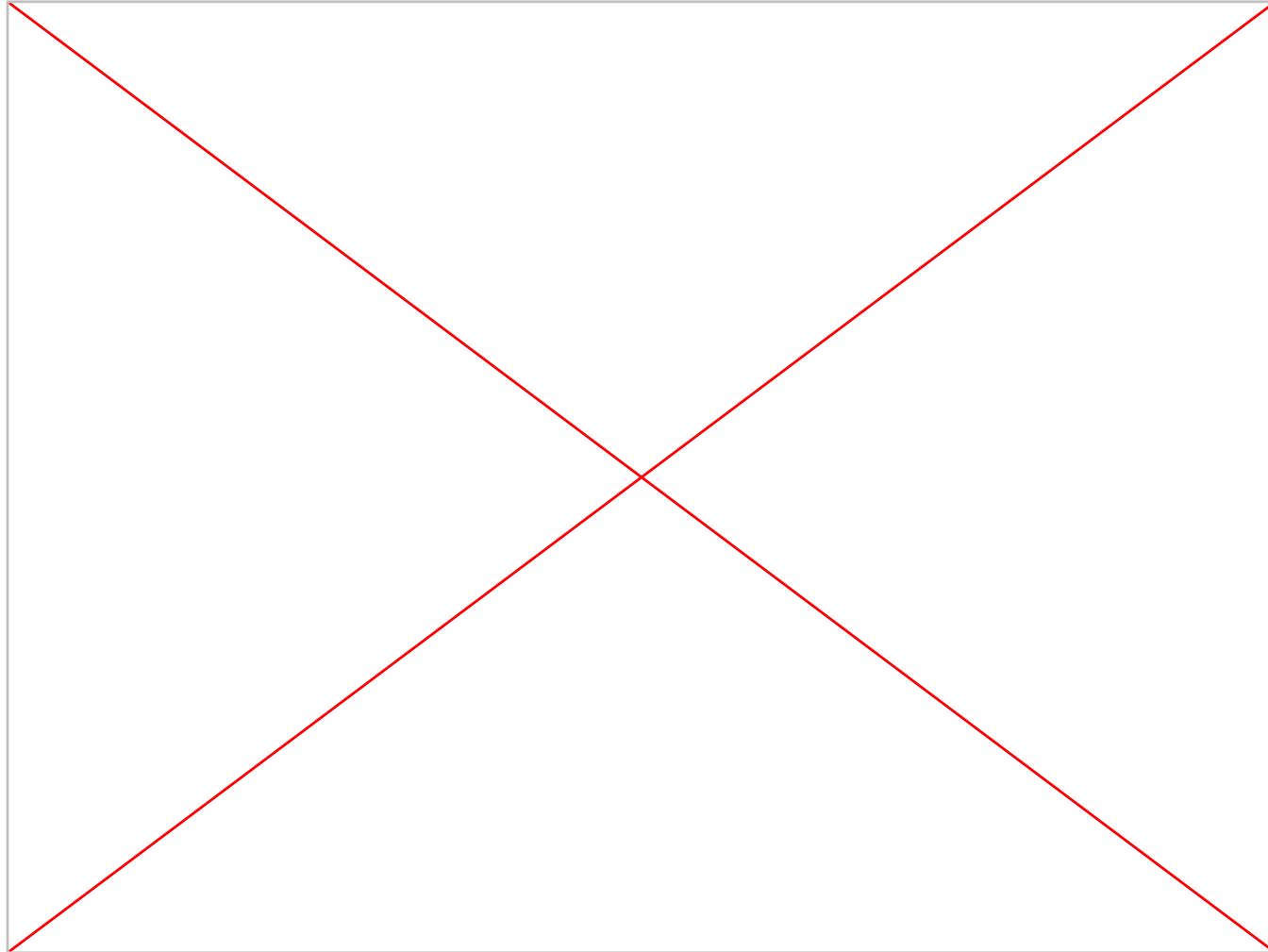
Runs on CPU/GPU



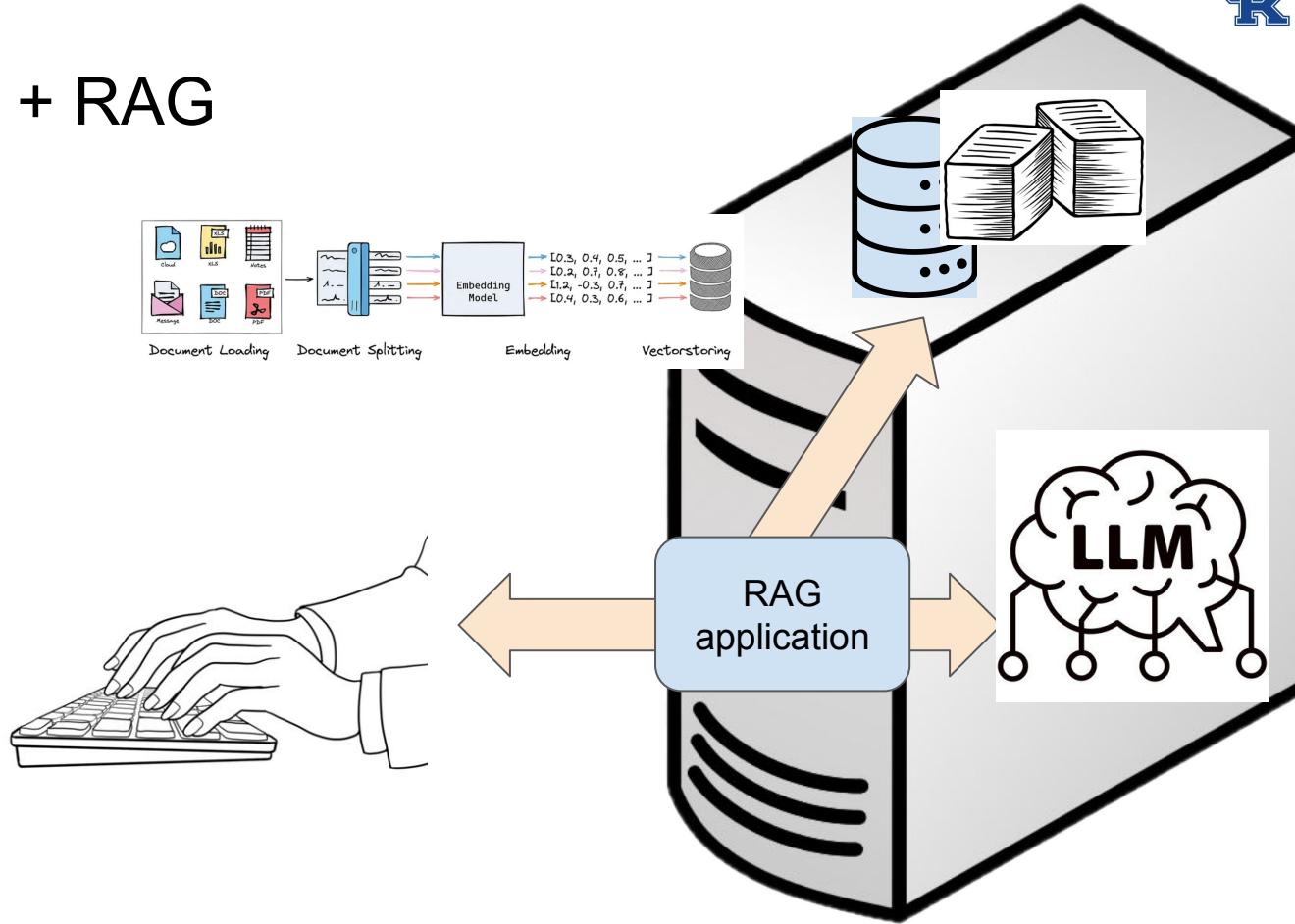
```
systemctl start ollama.service
ollama run <LLM model>
```

Running Ollama

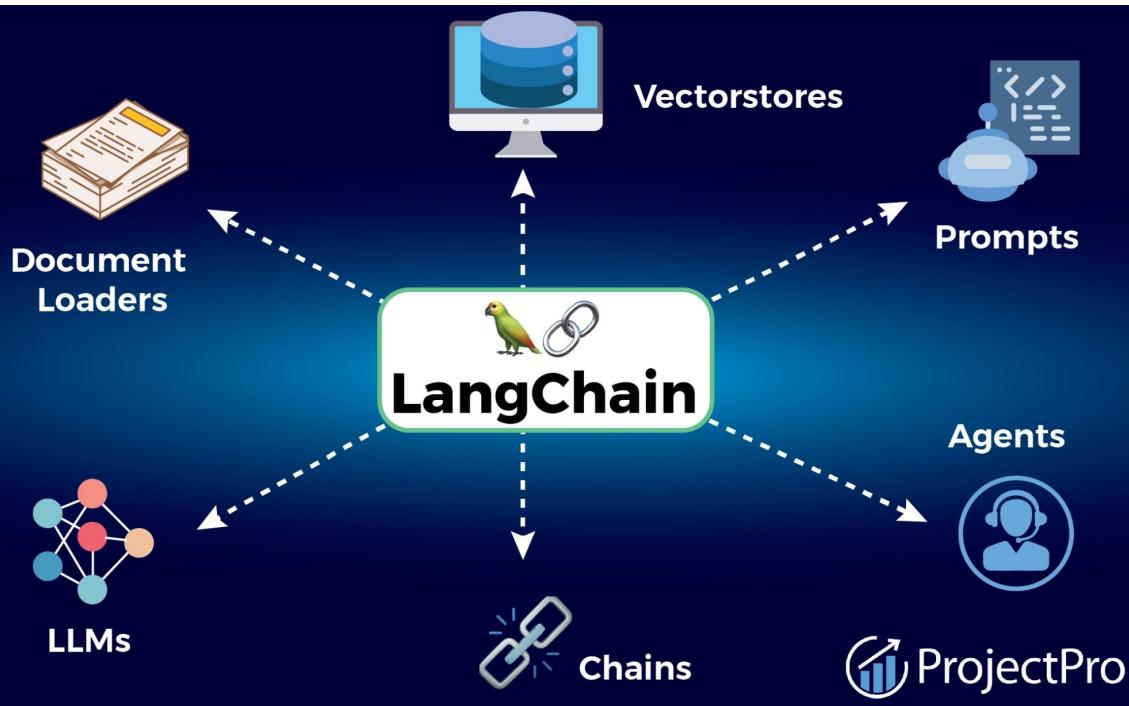
```
[mhaya2@gh3-internal ~]$ ollama list
NAME                  ID          SIZE      MODIFIED
mistral-large:latest  bbcf36dc47ad  73 GB    2 weeks ago
codestral:latest       0898a8b286d5  12 GB    2 months ago
codegemma:7b           0c96700aaada  5.0 GB   2 months ago
llama3.1:8b            91ab477bec9d  4.7 GB   5 months ago
codellama:13b          9f438cb9cd58  7.4 GB   6 months ago
llama3.1:70b           073e22d7e65d  39 GB    6 months ago
codellama:latest        8fdf8f752f6e  3.8 GB   6 months ago
codellama:70b          e59b580dfce7  38 GB    6 months ago
[mhaya2@gh3-internal ~]$
[mhaya2@gh3-internal ~]$
[mhaya2@gh3-internal ~]$ ollama run llama3.1:8b
>>> Send a message (/? for help)
```



LLM + RAG



Langchain



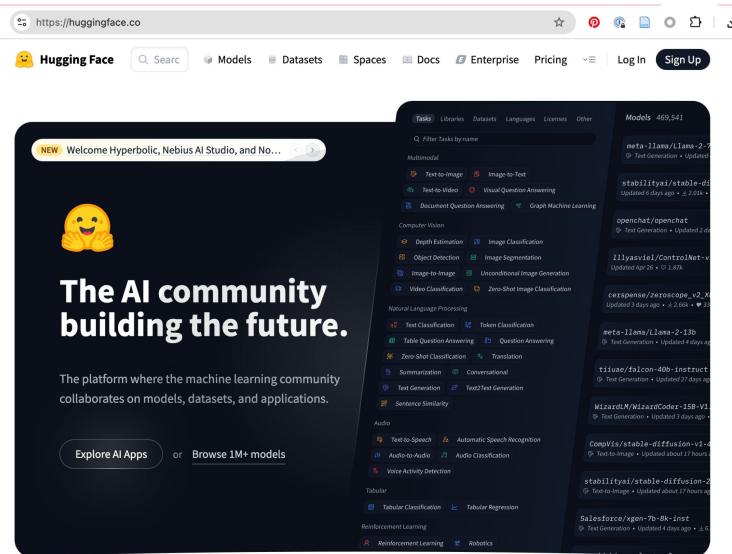
LangChain is a framework designed to simplify the creation of applications using large language models

Hugging Face



Hugging Face

- “GitHub for AI”
- Platform for sharing datasets, models etc.
 - Example: <https://huggingface.co/models>
- Keeping up with the latest/hottest
 - <https://huggingface.co/spaces/mteb/leaderboard>

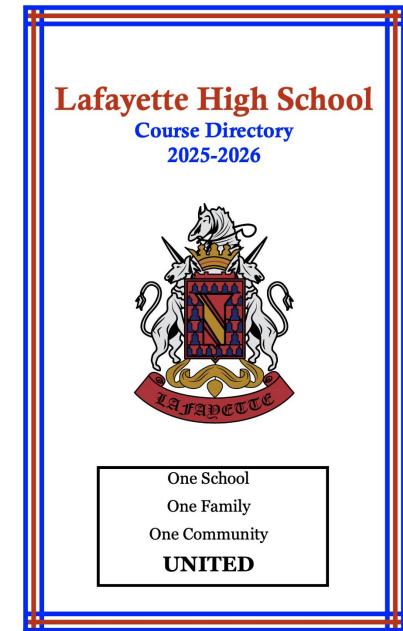


The screenshot shows the Hugging Face homepage with a dark theme. At the top, there's a navigation bar with links for Libraries, Datasets, Languages, Licenses, Other, and a search bar. Below the navigation, a large banner features a yellow emoji of a smiling face with hands clasped, followed by the text "The AI community building the future." and "The platform where the machine learning community collaborates on models, datasets, and applications." Below the banner, there are sections for "Explore AI Apps" and "Browse 1M+ models". The main content area lists various AI models and their categories, such as Multimodal, Computer Vision, Natural Language Processing, Audio, Tabular, Reinforcement Learning, and Robotics. Each model entry includes a thumbnail, name, and a brief description.

<https://huggingface.co/>

(Very simple) local RAG demo/tutorial

- Interact with Ollama using python-ollama
- Interact with Ollama using Langchain
- Create a Chroma vector database from a PDF file
- Run a query on RAG pipeline using Langchain, Ollama, and Chroma DB



<https://lafayette.fcps.net/>

Demo notebooks

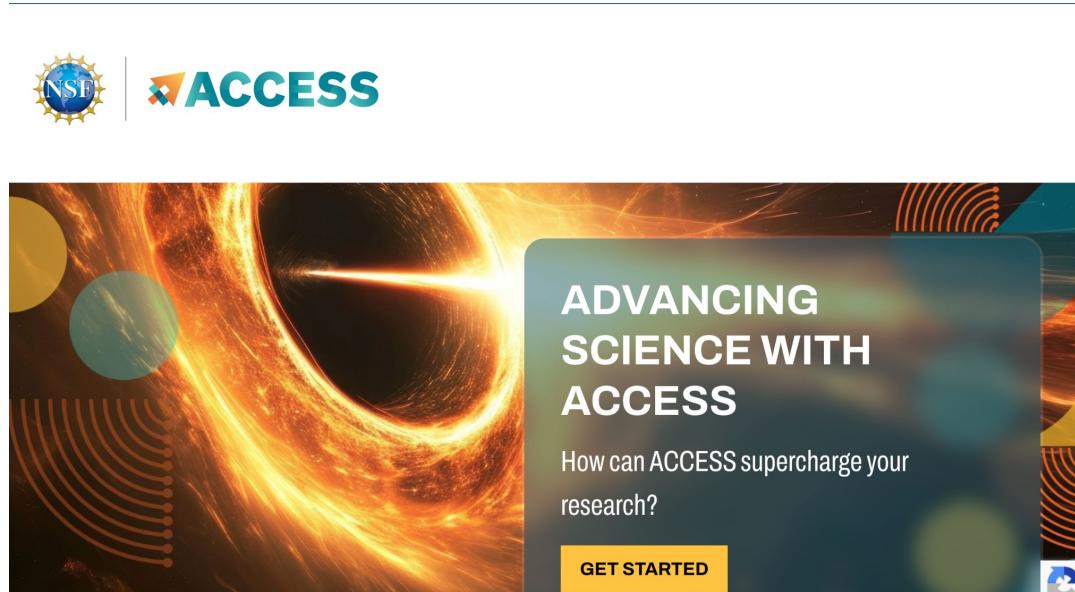
https://github.com/UKY-CCS-ITS-RCI/Workshops/tree/main/2025-02-20_ai_seminar



RAG Application Examples

1. ACCESS-CI

<https://access-ci.org/>



The banner features the NSF logo and the ACCESS logo. The background is a vibrant, abstract illustration of scientific concepts like energy, motion, and data. A central dark blue box contains the text "ADVANCING SCIENCE WITH ACCESS" and a question "How can ACCESS supercharge your research?". At the bottom is a yellow "GET STARTED" button.

NSF | ACCESS

ADVANCING
SCIENCE WITH
ACCESS

How can ACCESS supercharge your
research?

GET STARTED

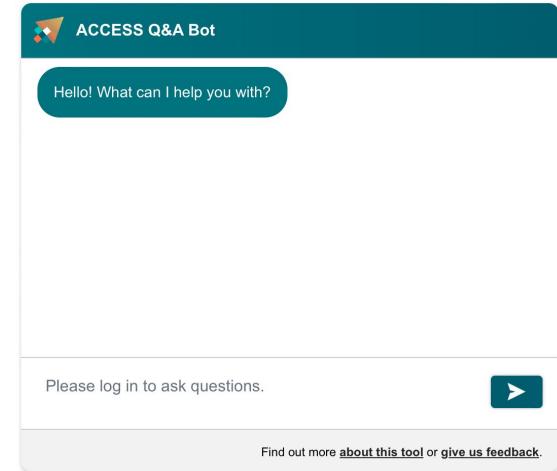
Introducing ACCESS-CI

- What is ACCESS-CI?
 - Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (<https://access-ci.org/>)
 - Connects researchers to computational resources in the U.S.
- Key Features:
 - Diverse computing resources, Data and storage services, Scientific applications, workflow management, science gateways
 - Comprehensive user support, resources for your class, training, and workshops, as well as connecting with communities that share your interests and learn from one another(affinity groups).

Q&A Tool

Providing information about the ACCESS ecosystem

- AI-driven information retrieval system
- Retrieval-Augmented Generation (RAG) and LLMs
- Added to Support portal in Sept 2024
(<https://support.access-ci.org>)
- Not a chatbot—each query is independent
- Reviewing questions, answers, and feedback to assess performance
- Developing a process to manage assessment, data sources, and cadence for data updates

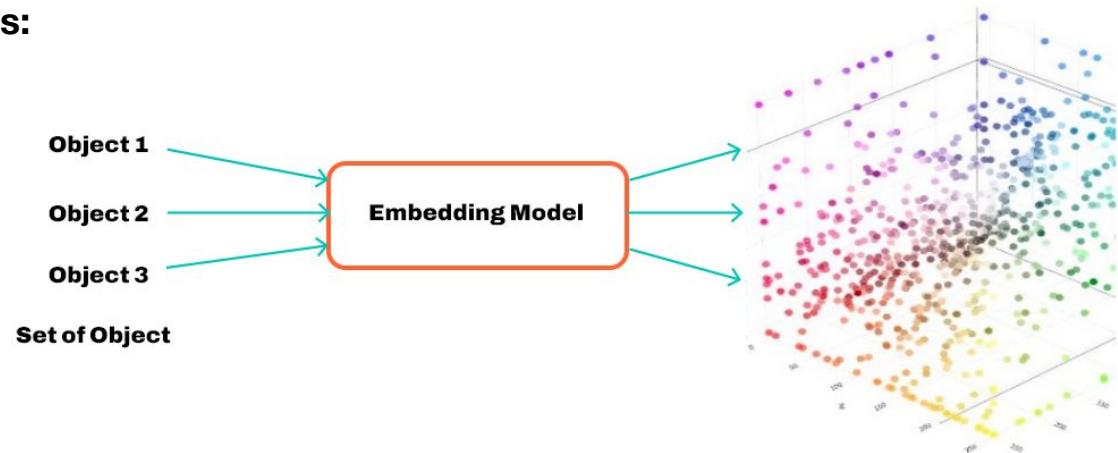


Data Ingestion

141 data sources indexed from ACCESS and RP sources

Database of embeddings sources:

- Data source url
- Maintaining team/track
- Description
- Keywords
- Embedding date

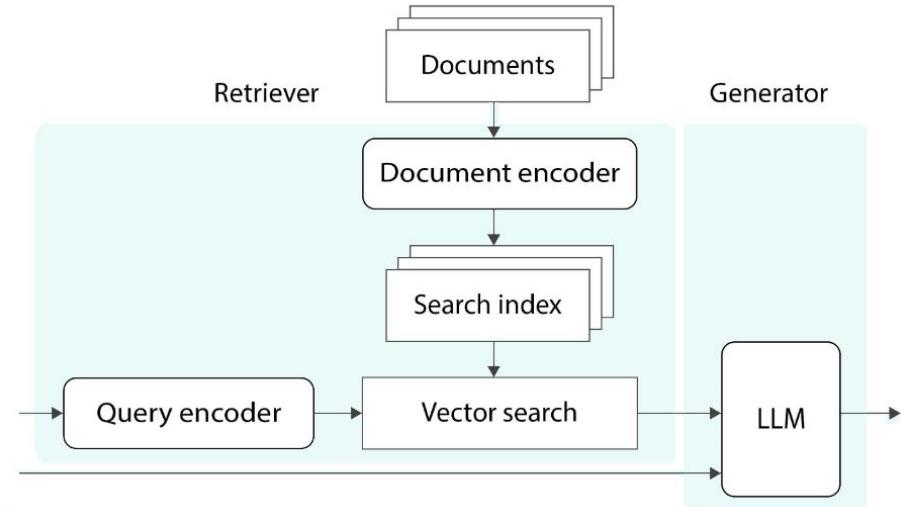


RAG LLM

Technologies used

- OpenAI for LLM inferencing
 - [GPT-4o-mini](#) in production
- Document embeddings use [OpenAI models](#)
 - text-embedding-3-small in production
 - testing out text-embedding-3-large
- Vector stores database
 - [ChromaDB](#) in production
 - Testing [Qdrant](#) and [FAISS](#) for expansion/scaling purposes
- [llamaindex](#) framework to integrate components

These technologies are used with extensive data curation and tailored metadata to enhance and improve the tool's responses.



AI Agentic Capabilities:

- Enhances structured retrieval processes.
- Filters and categorizes responses based on content metadata.
- Ensures dynamic adaptability to new data sources without compromising accuracy.

Key Considerations for Effective RAG Implementation (1):

Vector Databases, Chunking and Metadata in RAG

Vector Databases store document embeddings for efficient similarity search. Selecting the right vector database and indexing method affects retrieval accuracy and speed.

Chunking breaks long documents into smaller sections (256-512 tokens per chunk) before embedding.

Each chunk is embedded into a high-dimensional vector (e.g., 1536D for OpenAI embeddings).

Why Chunking? Smaller chunks improve search relevance by focusing on specific context.

Metadata storage: Store **document titles, keywords, and section headers** for better filtering.

Key Considerations for Effective RAG Implementation (2):

Vector Quantization vs. LLM Quantization

Vector Quantization (Used in Vector Databases & RAG)

Purpose: Reduce storage and speed up similarity search for embeddings.

Method: Compresses document embeddings (e.g., FP32 → INT8).

Impact: Enables fast retrieval in **ChromaDB, FAISS, Pinecone**.

LLM Quantization (Used for Model Inference & Deployment)

Purpose: Reduce LLM size to run efficiently on CPUs/GPUs.

Method: Compresses model weights (e.g., FP32 → INT4).

Impact: Enables Llama 3, GPT-4, Mistral to run on edge devices.

Key Difference:

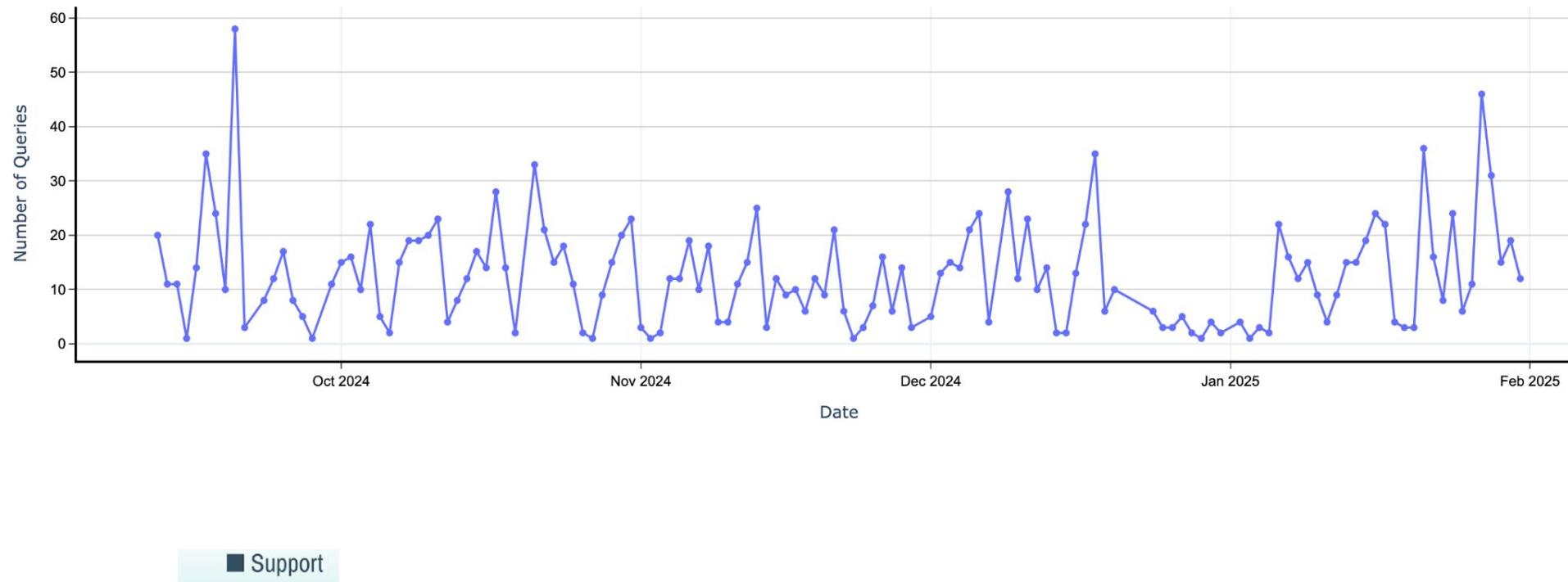
Vector quantization helps RAG search faster.

LLM quantization speeds up model inference.

Quantization is a technique that reduces the precision of numerical data (e.g., embeddings or model weights) to optimize storage and speed while maintaining accuracy.

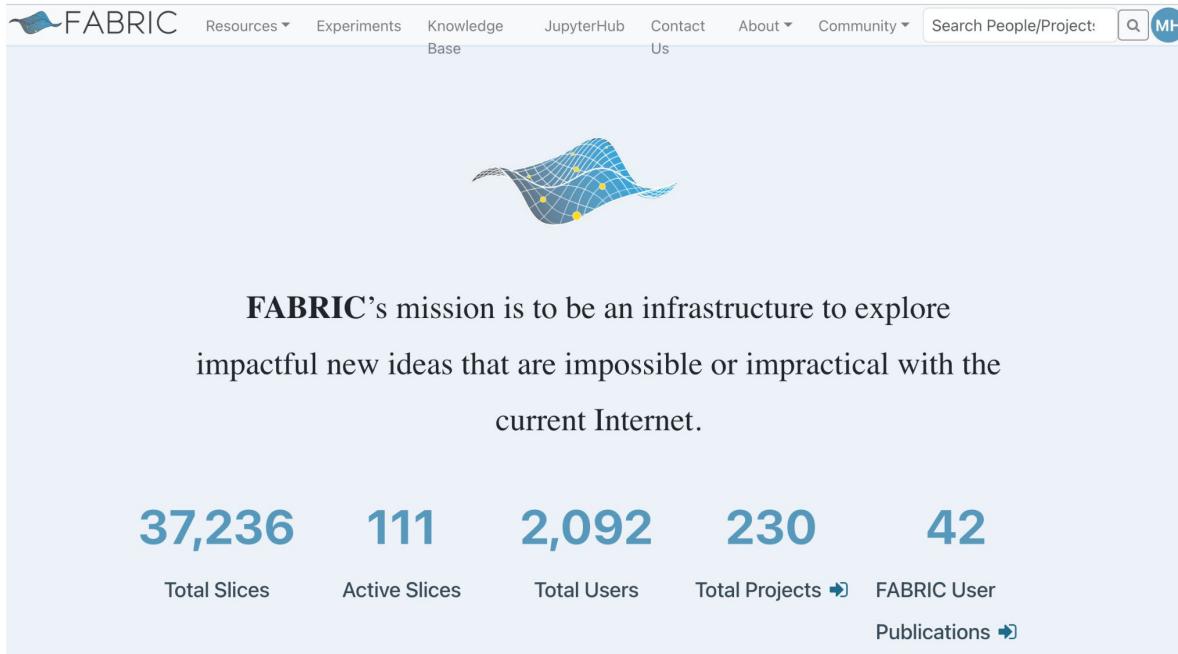
Q&A Tool

1822 queries since September 2024



2. FABRIC Testbed

<https://portal.fabric-testbed.net/>



The screenshot shows the FABRIC Testbed homepage. At the top, there is a navigation bar with links for Resources, Experiments, Knowledge Base, JupyterHub, Contact Us, About, Community, and a search bar for People/Project. Below the navigation bar is a large graphic of a wavy surface with yellow dots representing data points. The main text on the page reads: "FABRIC's mission is to be an infrastructure to explore impactful new ideas that are impossible or impractical with the current Internet." Below this text are five large blue numbers: 37,236, 111, 2,092, 230, and 42, each with a corresponding label below it: Total Slices, Active Slices, Total Users, Total Projects, and FABRIC User Publications.

Total Slices	Active Slices	Total Users	Total Projects	FABRIC User Publications
37,236	111	2,092	230	42

FABRIC project

Setting up and running experiments require writing a python code
(notebooks encouraged)



Setup the Experiment

Import the FABlib Library

```
[ ]: from fabricatestbed_extensions.fablib import FablibManager as fablib_manager
fablib = fablib_manager()
fablib.show_config();
```

Create the Experiment Slices

Create the slice and set the specific node attributes. Note that the cores, ram, and disk are only *hints*. The actual values will be the closest instance type that is larger than the chosen values.

Amounts of cores, ram, and disk will be rounded up to the closest instance type. These amounts should be considered minimums rather than specific requirements. You may find [this article](#) useful in explaining how VM size allocation works in FABRIC.

Note that your project must have `VM.NoLimit` permission set in order to allocate VMs larger than 2 cores, 10G of RAM and 10G of disk. Otherwise you will receive an error of the type `PDP Authorization check failed - Policy Violation: Your project is lacking VM.NoLimitCPU or VM.NoLimit tag to provision VM with more than 2 cores.`

```
[ ]: slice_name = 'MySlice1'

#Create Slice
slice = fablib.new_slice(slice_name)

# Add node
node = slice.add_node(name='Node1',
                      site='MAX',
                      cores=4,
                      ram=16,
                      disk=100,
                      image='default_ubuntu_20')

#Submit Slice Request
slice.submit()
```

Alternatively, you can specify a specific instance type by name. A full list of available instance types is [here](#).

```
[ ]: available_images = fablib.get_image_names()

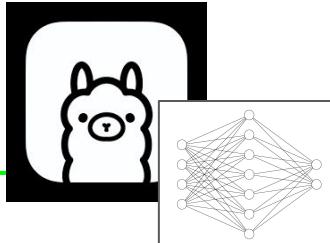
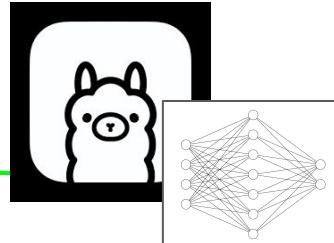
print(f'Available images are: {available_images}')

slice_name = 'MySlice2'
```

FABRIC code auto-generation using LLM + RAG



200+
Jupyter Notebooks



- Codellama
- Minstral
- Codestral
- Deepseek etc.

show me how
to set up a
slice with 3
nodes ...



```
Notes: breaks when trying to add component to the node. (node is not a component, so it has no "node" part, interface). This will tries to grab the interfaces of this specific component. This will give it a list of interfaces. And then it needs to grab the first one of them and add it to the L2 connection. But not only does this code not grab the first one but it also adds the whole list to the L2 network--using a "attach" function that doesn't exist.
```

```
from fabricated_extensions.fablib import FablibManager as fablib_manager

# Initialize the fablib manager
fablib = fablib_manager()

# Show the current configuration (optional)
fablib.show_config()

# Define slice name and node names
slice_name = 'mySlice'
node1_name = 'node1'
node2_name = 'node2'

# Get the random sites for the nodes
(sites1, sites2) = fablib.get_random_sites(count=2)

# Create a new slice
slice = fablib.new_slice(name=slice_name)

# Add Node1 to Slice
node1 = slice.add_node(node1_name, site=sites1)

# Add Node2 to Slice
node2 = slice.add_node(node2_name, site=sites2)
```

Questions?

Mami Hayashida (mami.hayashida@uky.edu)
Vikram Gazula (gazula@uky.edu)