Centre de Calcul
de l'Institut National de Physique Nucléaire
et de Physique des Particules

Proof-of-Concept of a chatbot based on the CC-IN2P3 documentation

FJPPN - February 11th, 2026

Contents

- Context and Objectives
- What about LLM ?
- What about RAG ?
- Source documentation : from RST to JSON
- Chunking and Embedding
- Running LLM
- Conclusion
- Next steps

Introduction

Context and Objectives

Context and Objectives

- Context:

  → Many tickets come from users who **don’t read** or **don’t understand** the CC documentation.

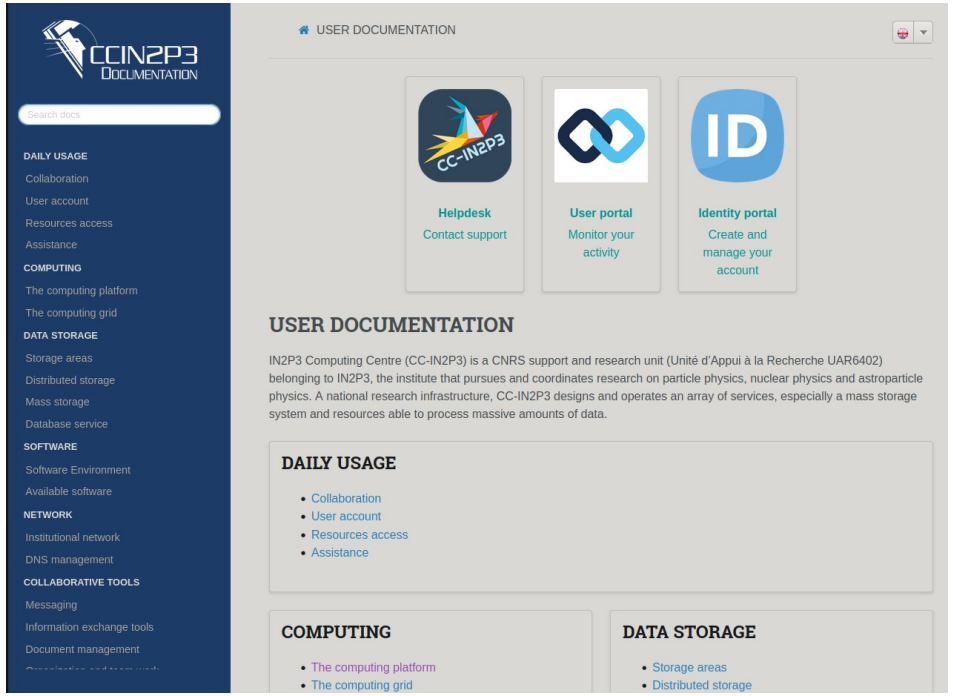
  → Waste of time for support teams.

- Objectives:

  A **RAG-LLM chatbot**

  → First-line support: automatically answer common questions.

  → Reduce ticket volume: free up time for complex issues.



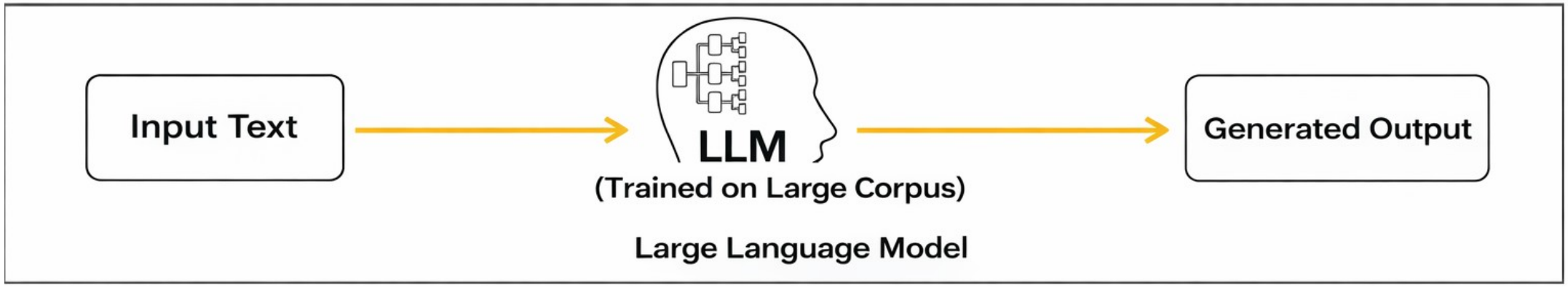
- Work produced by Mattéo Belz during a two-month internship at the CC in 2025.

What about LLM ?

What about LLM ?

LLM = Large Language Model

Deep neural networks trained on vast text corpora to generate human-like text.



Problems:

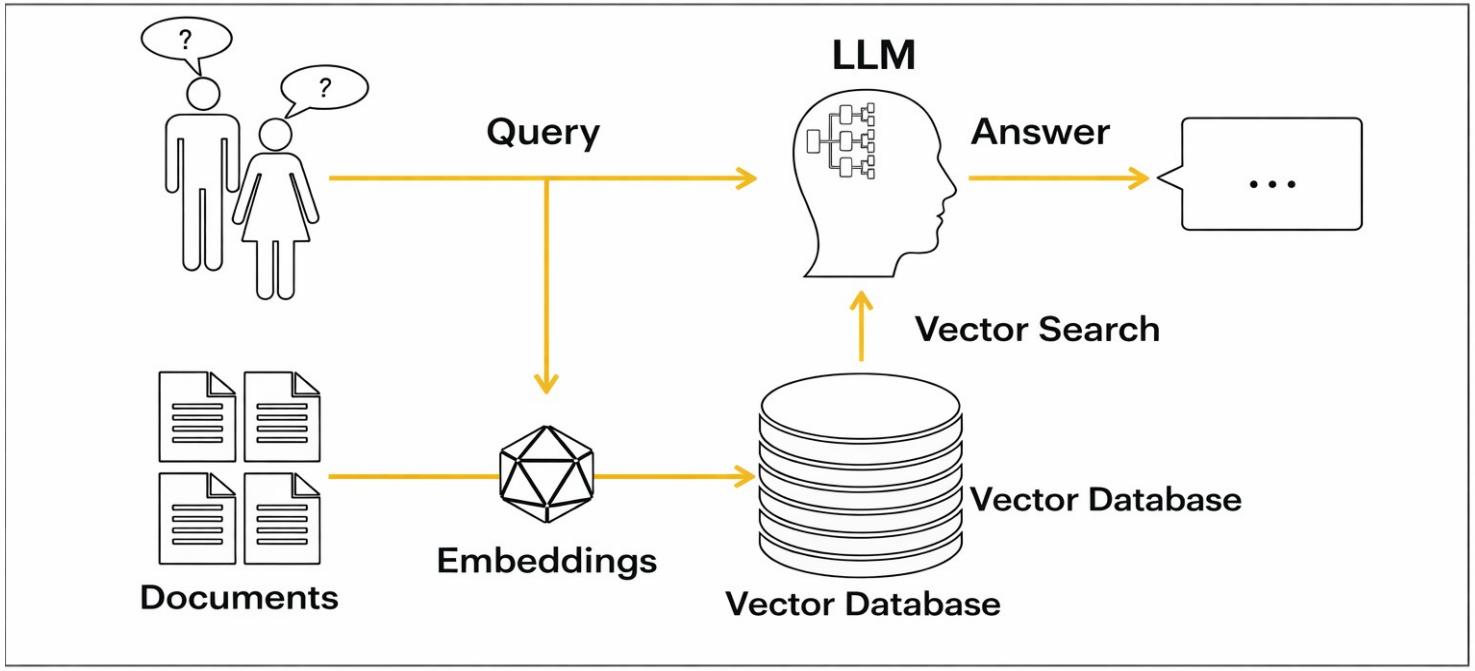
- limited knowledge (no updates after training)
- risk of hallucinations (made-up answers).

What about RAG ?

What about RAG ?

RAG = Retrieval-Augmented Generation

Combines documentation search (retrieval) with LLM (generation).
Update responses without retraining the LLM
Provides accurate, context-aware answers grounded in official docs.



Source Documentation

from RST to JSON

Source documentation : from RST to JSON

RST (ReStructuredText) :

A structured text format used for technical documentation

→ RST files mix tags, metadata, and raw text.

→ Not directly usable for RAG : needs clean, structured text.

```
=========
GPU Jobs
=========
To request a GPU, use:
.. code-block:: bash
   sbatch --gres=gpu:v100:1 job.sh
```

JSON (Key-Value Format) :

Remove RST tags (titles, code blocks, comments).

→ Preserve hierarchy (titles - subheadings – content).

→ Compatible with vector databases (e.g., ChromaDB).

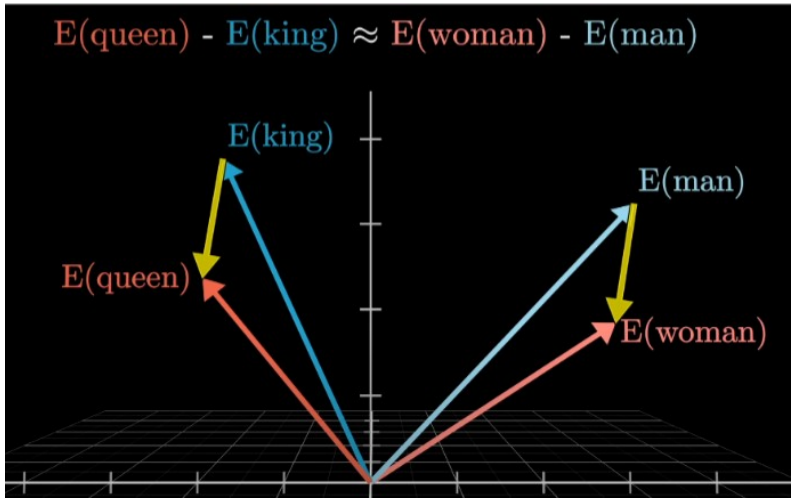
```
{
  "title": "GPU Jobs",
  "content": "To request a GPU, use: `sbatch --gres=gpu:v100:1 job.sh`",
  "language": "en",
  "category": "Computing"
}
```

Chunking and Embedding

Chunking and Embedding

- Chunking : Divide JSON into chunks (semantically coherent units).
  → Each chunk = a independent piece of information (e.g., a paragraph + its title).

- Vectorization : Numerical representations of text in a high-dimensional space.
  → Similar token have similar vectors.



- Cosine Similarity : Measuring similarity between vectors.
  → The closer the score is to 1, the more similar the vectors are.
  → In our case : *Score = 0.7 × similarity(title) + 0.3 × similarity(content)*

- ChromaDB : Open-source vector database, fast similarity search.
  → Useful because 2 collections : in French and in English.

Running LLM

Running LLM

- Ollama : Open-source, local execution, easy to deploy and manage, supports GPU acceleration.

  Questions/Answers test based on CC Documentation.

  → Model used : **Deepseek R1 32B.**

  → Alternatives tested: Llama 3 70B (too slow), Mistral 7B (less accurate).

- Parameters :

  → **Temperature:** Controls randomness (low (<=0) : precise, high (>=1) :creative).

  → In our case: 0.2

→ **Top-k:** We only keep the top-k tokens with the higher probablity.

  → In our case: 50

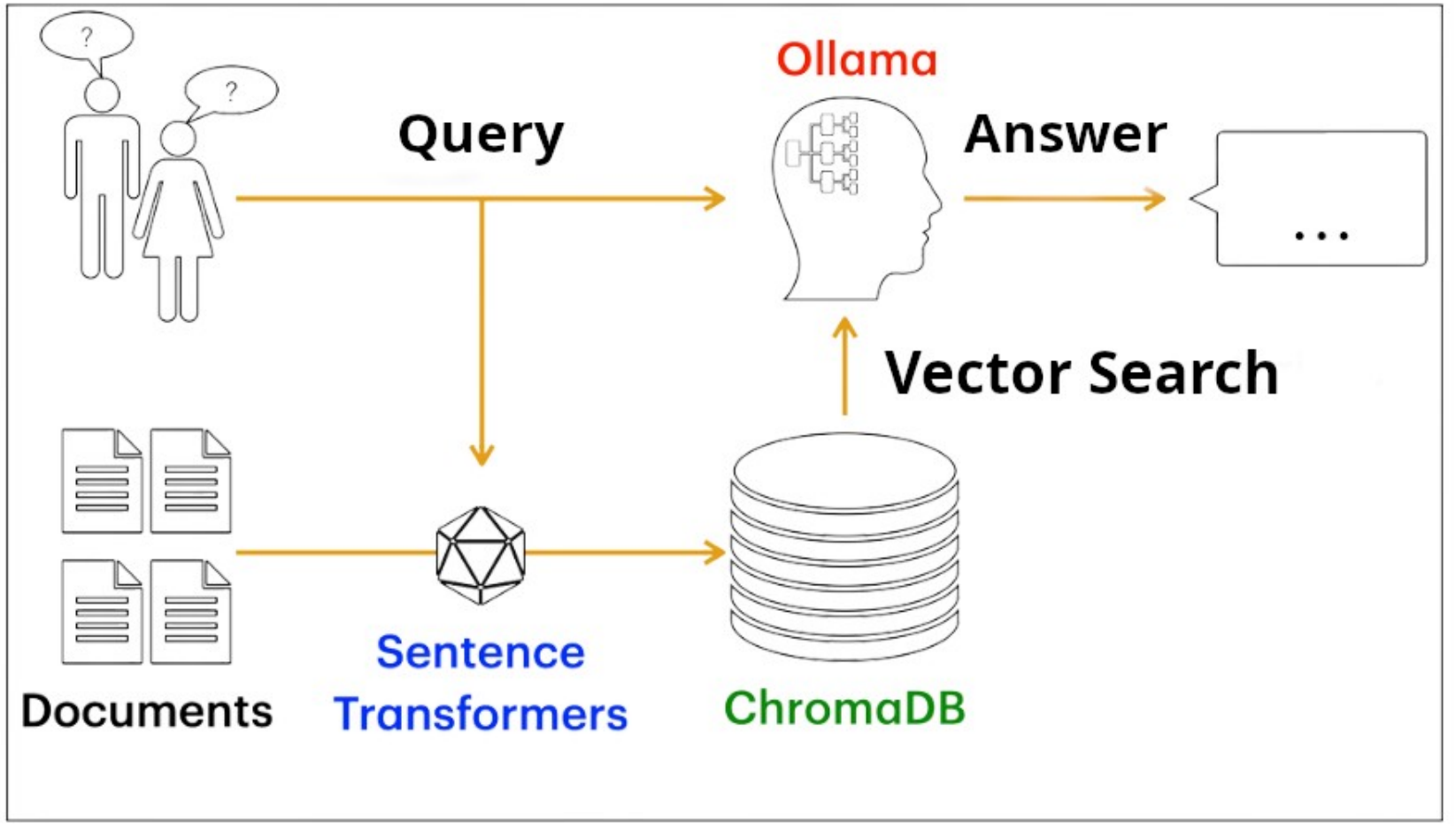
→ **Top-p**: Selects tokens until their cumulative probability reaches p (0 < p < 1).

  → In our case: 0.9

Conclusion

Conclusion

- Pipeline :



Next steps

Next steps

- Validate the information extracted from the RAG part (chunks + embedding part).
- Is the 70% title and 30% content ratio sufficient?
- Documentation enrichment (ticket examples, code…).
- Fine-tuning : Retrain the LLM on our data to improve accuracy.