

Agent conversationnel et chatbot

Utilisation de Large Language Models pour publier une
base de connaissances
Partie 4

Nicolas Debeissat
nicolas.debeissat@mail-formateur.net

Plan du module

Pourquoi faire un agent conversationnel ?

Comment défendre votre projet en entreprise ?

Principes de base des LLMs

Ce qui va faire qu'il va donner des bons résultats, ou non

Entraînement - Fine Tuning

Comment adapter ses réponses

Retrieval Augmented Generation

Répondre à partir de vos données

Intégration dans une application

Créer une API à partir de votre modèle

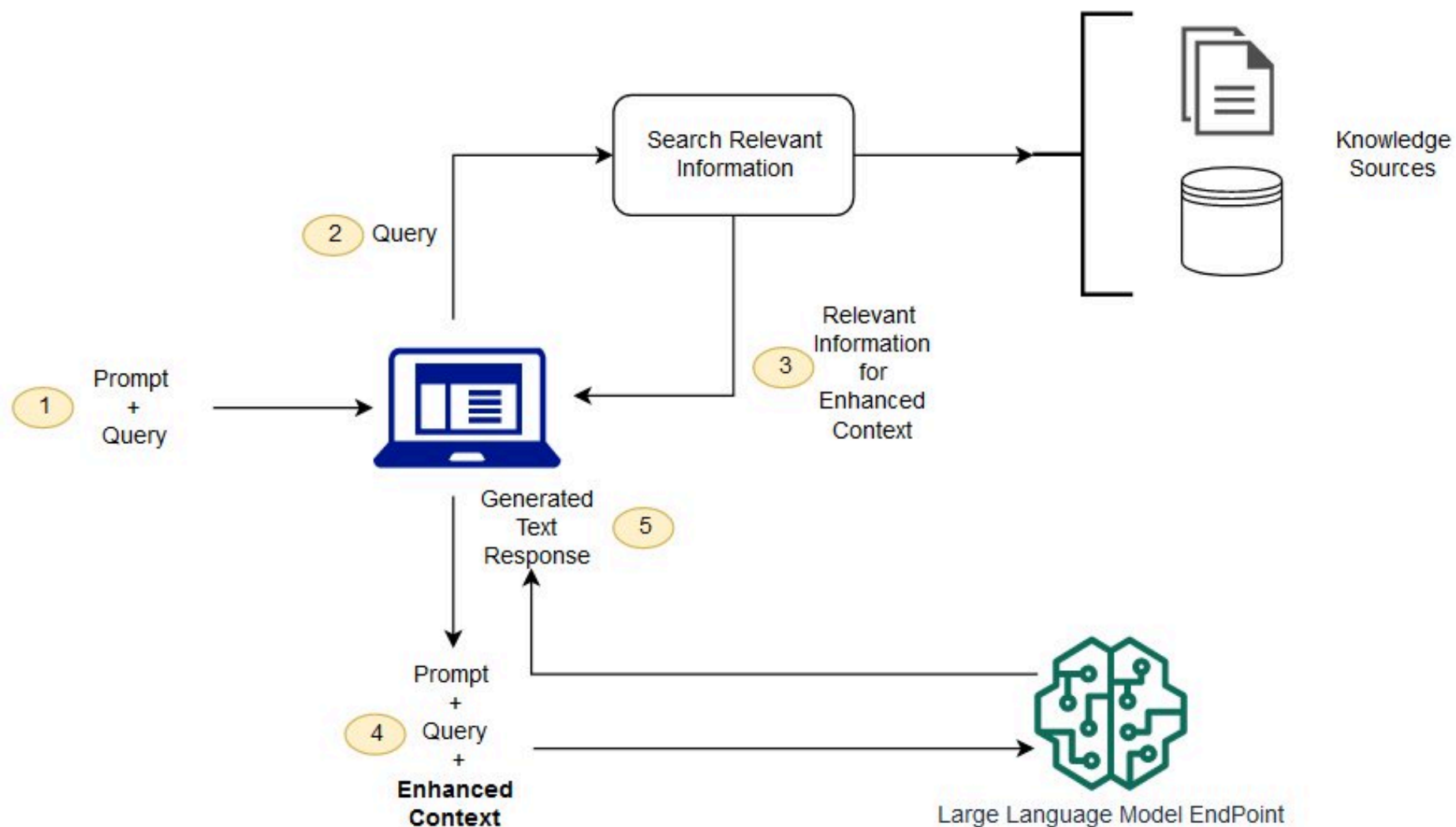
Evaluation

Entraîner votre IA à faire votre évaluation

Retrieval Augmented Generation

Répondre à partir de vos données

Principe du RAG



- Recherche dans la base de connaissances des documents les plus appropriés pour répondre à la question.
- Passage dans le prompt pour le LLM du contenu des documents récupérés en lui demandant de répondre à la question en utilisant ces textes.

Few shots learning + structured output

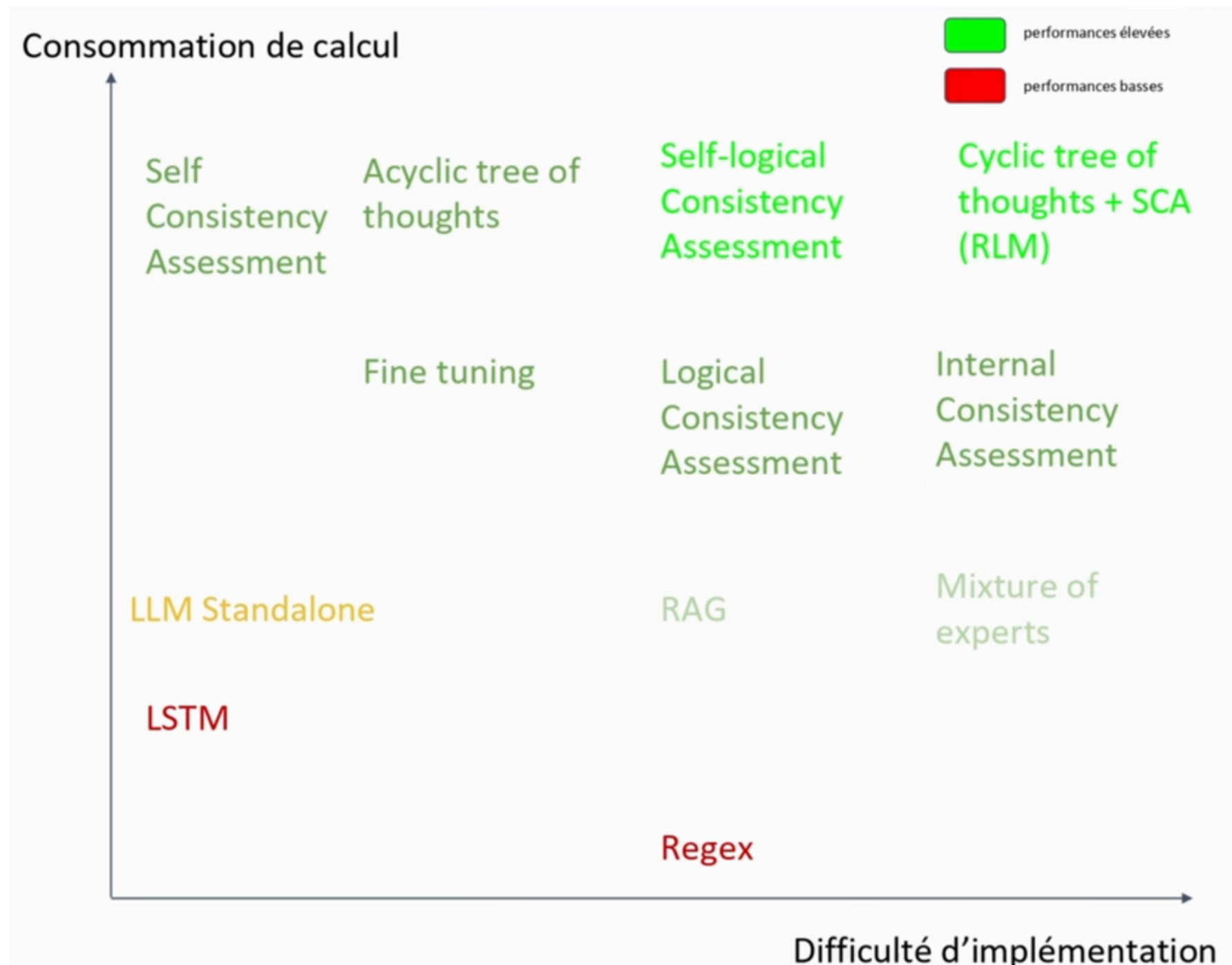


Prompt Engineering - Bonnes pratiques

```
{
  "context": "You are a medical decision-support assistant used in a hospital environment. Your role is to analyze medical documents and extract precise information regarding the patient's smoking habits. The goal is to assist physicians in documentation and harmonization of medical records while minimizing any misinterpretation.",
  "task": "Analyze the provided medical document and determine whether the patient is a smoker, non-smoker, or former smoker. Base your answer exclusively on explicit information found in the document. If no explicit mention is made, respond with 'unknown'. Always justify your answer by explaining the reasoning behind your classification.",
  "format": {
    "output_schema": {
      "smoking_status": "string (possible values: 'smoker', 'non-smoker', 'former smoker', 'unknown')",
      "reasoning": "string (detailed explanation of the reasoning process)"
    }
  },
  "examples": [
    {
      "input": "The patient quit smoking five years ago after consuming one pack per day for 20 years.",
      "output": {
        "smoking_status": "former smoker",
        "reasoning": "The document explicitly states that the patient smoked in the past and quit five years ago."
      }
    },
    {
      "input": "There is no mention of tobacco use in the patient's medical history.",
      "output": {
        "smoking_status": "unknown",
        "reasoning": "The document does not contain any information regarding the patient's smoking status."
      }
    }
  ],
  "input": "<medical_document_here>"
}
```

- Prompt système : indique un contexte général, le ton à adopter, le degré de complexité à donner
- instruction : la tâche à réaliser
- Few shots learning : ajout dans le prompt d'exemples de questions-réponses
- structured output : ajout dans le prompt du schéma de la sortie désirée
- le document à analyser
- le raisonnement si le modèle le permet

Après le RAG



- Vérification du raisonnement suivi par le LLM
- Interrogation en chaîne du modèle pour valider les différentes étapes du raisonnement
- Mesure de l'incertitude en analysant les probabilités générées par le LLM
- etc...

TD4-1 : Déploiement LLM avec ollama

Déployer ollama sur son poste et l'interroger :

<https://anakin.ai/blog/how-to-install-ollama-on-windows/>

Télécharger l'installeur : <https://ollama.com/download/windows>

Pour utilisation à distance : créer une variable d'environnement

OLLAMA_HOST à 0.0.0.0

Démarrer zephyr : `ollama run zephyr`

```
curl http://localhost:11434/api/generate -s -d '{"model": "zephyr",  
"prompt": "explique-moi le word embedding"}'
```

puis

```
curl http://localhost:11434/api/generate -s -d '{"model": "zephyr",  
"prompt": "explique-moi le word embedding", "stream": false}'
```

TD4-2 : Déploiement exemple langchain

https://templates.langchain.com/?integration_name=sql-ollama

Installer poetry :

<https://python-poetry.org/docs/#installing-with-the-official-installer>

```
poetry new td-langserve
```

```
code td-langserve
```

```
poetry config virtualenvs.in-project true --local
```

```
poetry show -v
```

```
modifier : requires-python = ">=3.12,<4.0" dans pyproject.toml
```

```
poetry add langchain-cli
```

```
source .venv/Scripts/activate
```

```
langchain app new my-app
```

```
cd ./my-app
```

```
langchain app add sql-ollama --branch v0.2
```


TD4-2 : Déploiement exemple langchain suite

code .

```
poetry config virtualenvs.in-project true --local
```

```
poetry install
```

```
source .venv/Scripts/activate
```

```
poetry add httpx=0.27.2
```

dans server.py :

```
from sql_ollama import chain as sql_ollama_chain
```

```
add_routes(app, sql_ollama_chain, path="/sql-ollama")
```

```
langchain serve
```